Tutorial de Desplazamiento

Sumario

Ver los anteriores tutoriales básicos para más información sobre la creación básica de objetos, manejo del reloj, jerarquía de fotogramas, animaciones, cámaras & vistas, música & sonido, efectos(FXs) y física.

Este tutorial muestra como dibujar un paralaje por movimiento; parallax scrolling(EN).

Como puedes ver, no hay un código especial para el paralaje por movimiento en sí.
Actualmente, el plugin de rendereo por defecto de orx tendrá esto en cuenta por tí, dependiendo de como tu pongas los atributos de los objetos en el fichero de configuración.

Por defecto en este tutorial, el atributo `AutoDesplazamiento`(`AutoScroll`) está puesto en 'ambos'('both').
Esto significa que un paralaje por movimiento sucederá en ambos ejes X y Y cuando se muevan las cámaras.
Tú puedes probar a poner este valor a la X, Y o cuando lo remuevas.

Más allá del atributo `AutoScroll`, tu puedes encontrar `EscalaEnProfundidad`(`DepthScale`).
Este atributo es usado para automáticamente ajustar la escala de los objetos dependiendo en cuan lejos están de la cámara.
La cámara truncada más pequeña es, a la que más rápido se le aplicará esta autoescala.
Puedes probar y jugar con el posicionamiento de un objeto y los planos lejos & cerca de una cámara para lograr el desplazamiento deseado y frecuencias de escalado profundo que quieras.

Puedes cambiar la velocidad de desplazamiento (ej. la cámara se mueve rápido) en el fichero de configuración. Como es usual, puedes modificar su valor en tiempo real y hacer una recarga de la configuración histórica.

Como has podido ver, nuestra simple actualización de código mueve la cámara en el espacio 3D.
Presionando las flechas se moverá a traves de los ejes X y Y, y presionando las teclas control & alt a través de la Z.
Como dije anteriormente, todos los paralajes por movimiento sucederán porque los objetos han sido marcados apropiadamente.
Tu código meramente necesita mover tu cámara en tu escenario, sin tener ninguna molestia con los efectos de desplazamiento.
Esto te da un control total sobre tantos planos en desplazamiento quieras, y que objetos serán afectados por el.

El último punto concierne al cielo.
Como vimos en el tutorial de fotogramas, ponemos los fotogramas de los objetos como hijos de la cámara.
Esto significa que la posición puesta por el objeto cielo en el fichero de configuración siempre será relativo a la cámara.
En otras palabras, el cielo siempre seguirá la cámara.
Como pusimos ahí, por defecto, a la profundidad de 1000 (ej. el mismo valor de un plano truncado de cámara alejada), el se mantendrá en el fondo(background).

Detalles

As usual, we begin by loading our config file, getting the main clock and registering our `Update` function to it.
Lastly, we create our `Sky` background and all our `Cloud` objects.
Please refer to the previous tutorials for more details.

Now let's see our `Update` function. First, we get our camera speed from config and update it using to our DT so as not to be framerate dependent.

```
orxVECTOR vScrollSpeed;

orxConfig_PushSection("Tutorial");

orxConfig_GetVector("ScrollSpeed", &vScrollSpeed);
orxVector_Mulf(&vScrollSpeed, &vScrollSpeed, _pstClockInfo->fDT);

orxConfig_PopSection();
```

Nothing really new so far.
We now need to update our camera move vector depending on the active inputs.

```
if(orxInput_IsActive("CameraRight"))
{
  vMove.fX += vScrollSpeed.fX;
}
if(orxInput_IsActive("CameraLeft"))
{
  vMove.fX -= vScrollSpeed.fX;
}
if(orxInput_IsActive("CameraDown"))
{
  vMove.fY += vScrollSpeed.fY;
}
if(orxInput_IsActive("CameraUp"))
{
  vMove.fY -= vScrollSpeed.fY;
}
if(orxInput_IsActive("CameraZoomIn"))
{
  vMove.fZ += vScrollSpeed.fZ;
}
if(orxInput_IsActive("CameraZoomOut"))
{
  vMove.fZ -= vScrollSpeed.fZ;
}
```

Lastly we apply this movement to our camera.

```
orxCamera_SetPosition(pstCamera, orxVector_Add(&vPosition,
orxCamera_GetPosition(pstCamera, &vPosition), &vMove));
```

As stated before, there's not even a single line of code to handle our parallax scrolling.
Everything will be done on the config side. We simply move our camera in our 3D space.

Let's have a look at the config data. First our `Tutorial` section where we have our own data.

```
[Tutorial]
CloudNumber = 1000
ScrollSpeed = (300.0, 300.0, 400.0)
```

As you can see, we have our `ScrollSpeed` and our `CloudNumber` to control this tutorial.
The `ScrollSpeed` can be update on the fly and reloaded with the config file history (by pressing `Backspace`).

Let's now see our cloud object.

```
[CloudGraphic]
Texture = ../../data/scenery/cloud.png
Pivot   = center

[Cloud]
Graphic     = CloudGraphic
Position    = (0.0, 0.0, 100.0) ~ (3000.0, 2000.0, 500.0)
AutoScroll  = both
DepthScale  = true
Color       = (180, 180, 180) ~ (220, 220, 220)
Alpha       = 0.0
Scale       = 1.0 ~ 1.5
FXList      = FadeIn
```

The two important attributes here are `AutoScroll` that activates the parallax scrolling and `DepthScale`.

First, the `AutoScroll` attribute can take the values 'x', 'y' or 'both'.
This tells on which axis the parallax scrolling will happen for this object.
The parallax scrolling will be rendered accordingly to the object's Z coordinate (ie. its depth in the camera frustum).
The closest the object is to the camera on the Z axis, the faster it will scroll. `AutoScroll` default value is `none`.

The `DepthScale` attribute tells the render plugin wether to scale the object or not accordingly to its Z coordinate.
The closest the object is to the camera on the Z axis, the largest it will be displayed. `DepthScale` default value is `false`.

Let's now have a look at our sky object.

```
[SkyGraphic]
Texture = ../../data/scenery/sky.png
```

```
Pivot    = center

[Sky]
Graphic        = SkyGraphic
Scale          = (0.5, 0.004167, 1.0)
Position       = (0.0, 0.0, 1.0)
ParentCamera   = Camera
```

As you can see, we set a `ParentCamera` for our `Sky` object, meaning our `Sky` will be in `Camera`'s local space (ie. it will move along the camera).
We set it's position to `(0.0, 0.0, 1.0)` which means it's centered in the camera space and in the complete background.
When having a `ParentCamera`, `Scale` and `Position` are expressed in parent's space by default, unless explicitely refused by setting `UseParentSpace` to `false`.
Hence our 'weird' value for the scale. If we had an object made of a single pixel, a scale of (1.0, 1.0, 1.0) would cover the entire parent camera's view.
As our `sky.png` bitmap is 2 pixel wide on X axis, we need a scale on X of 0.5.

In the same way, as it is 240 pixel long on Y axis, we need a scale on Y of 1/240 = 0.004167. 😀

Recursos

Código fuente: 09_Scrolling.c

Fichero de configuración: 09_Scrolling.ini