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

Como es usual, comenzaremos por cargar nuestro fichero de configuración, obteniendo el reloj principal y registrando nuestra función `Update` a el.
Por último, creamos nuestro fondo `Cielo`(`Sky`) y todos nuestros objetos `Nube`(`Cloud`).
Por favor referirse a los tutoriales previos para más detalles.

Veamos ahora nuestra función `Update`. Primero, obtenemos la velocidad de nuestra cámara del fichero de configuración y la actualizamos usando a nuestro DT para que no sean dependientes las imágenes por segundo.

```
orxVECTOR vScrollSpeed;

orxConfig_PushSection("Tutorial");

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

orxConfig_PopSection();
```

Nada realmente nuevo hasta el momento.
Necesitamos ahora actualizar el vector de movimiento de nuestra cámara dependiendo en las entradas activas.

```
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;
}
```

Por último apliquemos este movimiento a nuestra cámara.

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

Como se dijo antes, no hay ni una sola línea de código para controlar nuestro Paralaje por movimiento.
Todo está hecho en la parte de configuración. Simplemente movemos nuestra cámara en nuestro espacio 3D.

Echemos un vistazo a los datos de configuración. Primero nuestra sección `Tutorial` donde tenemos nuestros propios datos.

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

Como puedes ver, tenemos nuestra `VelocidadDeDesplazamiento`(`ScrollSpeed`) y nuestro `NúmeroDeNubes` (`CloudNumber`) para para controlar este tutorial.

Veamos ahora nuestro objeto nube.

```
[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
```

Los dos atributos importantes aquí son `DesplazamientoAutomático`(`AutoScroll`) que activa el paralaje por movimiento y `EscalaProfunda`(`DepthScale`).

Primero, el atributo `AutoScroll` puede tomar los valores 'X', 'Y' o 'ambos'.
Esto dice en que eje está el desplazamiento por paralaje esta ocurriendo para este objeto.
El desplazamiento por paralaje será rendereado acorde a la coordenada 'Z' del objeto (ej. la profundidad de la cámara truncada).
El objeto más cercano a la cámara en el eje 'Z', será el que más rápido se desplace. El valor por defecto del `AutoScroll` es `ninguno`(`none`).

El atributo `DepthScale` le dice al plugin de render wether(?) como escalar el objeto o no en dependencia de su coordenada 'Z'.
El objeto más cercano a la cámara en el eje 'Z', será mostrado más grande. El valor por defecto de `DepthScale` es `false`.

Veamos nuestro objeto cielo.

```
[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
```

Como puedes ver, ponemos una `CámaraPadre`(`ParentCamera`) para nuestro objeto `Sky`, queriendo decir que `Sky` será el espacio local en nuestra `Camera` (ej. se moverá a largo de la cámara).
Ponemos su posición en `(0.0, 0.0, 1.0)` que significa que está centrado en el espacio de la cámara y en todo el fondo.
Disponemos de una `ParentCamera`, `Scale` y `Position` siendo expresados en padres del espacio por defecto, a menos que explícitamente se negaron poniendo `UseParentSpace` a `false`. De ahí nuestro `extraño` valor para la escala. Si tenemos un objeto hecho de un solo pixel, la escala de (1.0, 1.0, 1.0) pudiera cubrir la vista completa de la cámara padre.
Como nuestro `sky.png` mapa de bits(bitmap) es de 2 pixels de ancho en el eje X, necesitamos una escala en X de 0.5.
De la misma manera, como el mismo es de 240 pixels de largo en el eje Y, necesitamos una escala en Y de 1/240 = 0.004167. 😀

Recursos

Código fuente: 09_Scrolling.c

Fichero de configuración: 09_Scrolling.ini

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

Permanent link:
https://www.orx-project.org/wiki/es/orx/tutorials/scrolling?rev=1330962528

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