视差滚动(scrolling)教程

综述

参看前面的教程基础, 对象创建, 时钟 , 框架层次结构 动画 视口与摄像机 声音与音乐 特效
此教程说明了如何去展示一个 视差滚动(parallax scrolling)

正如你可以看到的，视角切换本身没有用到特别的代码。
事实上orx的默认2D 渲染插件将根据你在配置文件中怎么设置对象的属性来帮你设置好视差滚动。

By default, in this tutorial, the attribute AutoScroll is set to 'both'. 默认情况下，在这个教程里，配置AutoScroll将被设置为'both'表示两个坐标轴方向上都有滚动。
这意味着当摄像机移动的时候视差滚动将在X,Y两个坐标轴上同时发生。
你可以尝试设置这个值为x,y甚至可以删掉此配置值。

Along the AutoScroll attribute, you can find the DepthScale one.
除了AutoScroll的属性外，你可以找到DepthScale属性（深度缩放比）。 这个属性将用来依据对象离摄像机的距离自动的调整对象的缩放。
摄像机视锥体越小，自动缩放将会应用的越快。
你可以尝试将对象定位到摄像机的远&近剪裁平面来获取你感觉合适的视差滚动和深度缩放比。
你可以通过配置文件来改变视差滚动的速度（注释：摄像机移动的速度）。就像平时一样，你可以实时修改它的值并且重新读取配置。

正如你所能看到的，我们的update代码需要简单地移动3D空间里的摄像机就行了。
按下方向键能把摄像机视角按照X轴和Y轴移动，按下control和alt键会让视角按照Z轴移动。
正如前面说的，所有的视差滚动效果将会正常发生，因为所有的对象都已近被合适地处理了。[1)]

在场景里移动摄像机只需要很少的代码，而且不用去关心任何的滚动效果。[2)] 你对你想要有多少个滚动平面，以及哪些对象应该被滚动所影响有完全的控制。

最后一点与天空的显示有关。
正如框架教程(frame tutorial)里我们所看到的，我们把天空对象的frame设置为摄像机视角的一个子frame

这意味着在配置文件里对天空对象位置的设置将总是相对与摄像机位置的相对值。
换句话说，天空将总是跟随着摄像机视角。
当我们把它的深度值设置为默认值1000时，（注释：与视锥矩形的远剪裁平面的值相同）它将会停留在背景中。

详细说明

As usual, we begin by loading our config file, creating a 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_SelectSection("Tutorial");

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

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. 😀

Resources

Source code: 09_Scrolling.c

Config file: 09_Scrolling.ini

1)
译者注：包括缩放和位置
2)
译者注：即这些效果已经自动的实现了

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

Permanent link:
https://www.orx-project.org/wiki/cn/orx/tutorials/scrolling?rev=1278563787

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