本篇由tarzan_sp翻译自 官方的教程[1)]

卷轴效果(scrolling)教程

综述

参看前面的教程基础, 对象创建, 时钟 , 帧层次结构 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 视差卷轴(parallax scrolling) [2)]

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卷轴。

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卷轴效果将在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。

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卷轴速度和深度缩放比。
你可以通过配置文件来改变视差卷轴的速度（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，所有的视差卷轴效果将会正常发生，因为所有的对象都已近被合适地处理了。[3)]

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。[4)] 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

就像平时一样，我们通过加载配置文件，创建一个时钟，注册我们的Update函数。
最后，我们创建我们的天空背景和我们所有的云对象。
请回头看前面的教程来了解更多的细节。
现在，让我们来看Update函数。首先，我们从配置文件得到摄像机速度，然后通过依赖DT[5)]来更新它, 从而摆脱帧率依赖。

```
orxVECTOR vScrollSpeed;
```

```
orxConfig_SelectSection("Tutorial");

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

到目前为止没有什么是真正新鲜的。
现在我们需要根据输入改变摄像头移动的vector（向量）。

```
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;
}
```

最后我们将这个移动（向量）应用到摄像头上。

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

正如前面陈述的，我们将不需要写一行代码来控制视差卷轴。

所有的事将会在配置端被完成。我们简单地在我们的3D空间里移动我们的摄像头。

让我们看看配置数据。首先是Tutorial字段，用来存储我们自己的数据。

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

正如你所看见的，我们用ScrollSpeed和CloudNumber两个字段来控制这个程序。
ScrollSpeed可以在运行时修改然后重载读取配置文件（通过按Backspace键）进行更新。

现在让我们看看云朵对象

```
[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
```

激活视差卷轴的AutoScroll字段和DpthScale地段是非常重要的两个字段。
首先AutoScroll字段可以使用‘x’的值‘x’的值或者‘both’的值。
这将决定针对这个对象的视差卷轴效果将在哪个坐标轴上发生。
视差卷轴将根据对象的Z轴坐标（注释：它在摄像机的视锥矩形里面的深度）被渲染。
对象在Z坐标轴上距离摄像机越近，视差卷轴滚动进行得越快。
AutoScroll的缺省值是none\

DepthScale深度缩放比）属性将决定渲染插件是根据它的Z轴坐标来调整对象的比例。
对象在Z轴上距离摄像机越近，它将被显示得越大。
DepathScale深度缩放比）的缺省值是false

现在让我们看看我们的天空对象.

```
[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
```

正如你所看到的，我们为Sky对象设置了一个ParentCamera父摄像机），意味着我们的Sky将在摄像机的local space中（本地空间）（注释：它将跟随着摄像机一起移动）。

我们把天空的位置设为(0.0 ， 0.0 ， 1.0)，这意味着它被置于摄像机的视角的中央，同时它也成为摄像机所能看到画面的背景。

When having a `ParentCamera`, `Scale` and `Position` are expressed in parent's space by default, unless explicitely refused by setting `UseParentSpace` to `false`.

当有一个ParentCamera存在的时候Scale缩放比）和Position坐标）属性默认将在父对象的空间中表示，除非把 UseParentSpace使用父级空间）设置为false

因此,Sky对象的scale值在这里有些“奇怪”。

如果我们有一个对象是由单像素构成的，一个缩放比设置为（ 1.0 ， 1.0 ， 1.0 ）的对象将覆盖整个父级摄像机所能看到的画面。
因为我们的sky.png位图在X轴上的宽度是2像素，我们在X轴方向需要一个大小为0.5的 scale值。
同样的方式，因为它在Y坐标轴上的长度是240 像素，我们在Y轴上需要一个大小为1/240的scale(1/240 = 0.004167.)

资源

源代码：09_Scrolling.c

配置文件：09_Scrolling.ini

1)
九天注：按照同事的意思，将原来的视差滚动都按需求翻译成视差卷轴了，他认为那样更加合适
2)
happyfire注：就是多个背景，速度不同的卷轴，有立体感
3)
译者注：包括缩放和位置
4)
译者注：即这些效果已经自动的实现了
5)
译者注：指clock结构的DT字段，见下面的例子