本页由 落后的簑羽鹤 翻译自 官方的教程[]

视口与摄像机(viewport & camera)教程

综述

前面的基本教程基础, 对象创建, 时钟, 帧层次结构 和 动画[]

此教程显示了如何使用有多个摄像机的多视口技术。教程中将同时创建4个视口。

分别为左上角的[]Viewport1[]]右下角的[]Viewport4[]]它们共用一个摄像机(Camera1)[]实现此功能,只需要在配置文件中配置2个视口的Camera属性,为同一个(也就是Camera1)]]当我们使用鼠标的左右键旋转 摄像机(Camera1),left Control或left Shift键+方向键进行摄像机的缩放操作,关联的两个Viewport1 和Viewport4将相应的发生变化。

右上角视口[[Viewport2]]是基于另一个摄像机[]Camrea2[]]此摄像机的视锥较第一个窄,所以显示时比例是 其的两倍大。在教程的程序中,我们不能通过任何操作设置此视口。

最后一个视口[[Viewport3]]是基于Camera3的,Camera3的配置与Camera1完全一样。

NB[]当两个视口重叠,较先创建的将显示在顶层。

最后,有一个固定不动的箱子和一个世界坐标随着鼠标实时移动的小兵,也就是说无论如何设置视口的摄 像机,无论鼠标在那个视口上移动,小兵在它所属的视口中,相对于鼠标在在屏幕中的位置移动。

在配置文件中使用随机关键字符 '~',使的视口和基本对象的颜色和大小可以随机创建。

NB[]摄像机将它的坐标/缩放尺度/旋转存放在orxFRAME 结构中,在frame教程中我们看到他们 是orxFrame 继承体系的一部分。另一方面Object应该置于其Camera所关联的Viewport中。¹⁾

详细说明

常我们需要首先载入配置文件,创建时钟和注册回调的 Update函数,最后创建主要的Object信息。关于 实现的详情,请联系前面的教程。

虽然这次我们创建了4个视口,却没有什么新东西,仅仅是以下4行代码。

```
pstViewport = orxViewport_CreateFromConfig("Viewport1");
orxViewport_CreateFromConfig("Viewport2");
orxViewport_CreateFromConfig("Viewport3");
orxViewport_CreateFromConfig("Viewport4");
```

正如你所看到的,我们只使用了 Viewport1的引用,以便后面进行操作。

让我们直接跳到Update函数的代码。

首先我们通过捕捉鼠标的坐标,设置士兵的位置。我们已经在frame tutorial里实现过了。这里我们做了一样的事情,但在4个视口中工作的都很完美。当鼠标离开视口时,世界坐标的指针,将被orxNull值所代替,

Last update: 2018/02/14 00:46 (7 years ago) cn:orx:tutorials:viewport https://wiki.orx-project.org/cn/orx/tutorials/viewport?rev=1518583597

```
也就不会触发士兵的移动了。
```

```
orxVECTOR vPos;
```

if(orxRender_GetWorldPosition(orxMouse_GetPosition(&vPos), &vPos) !=
orxNULL)

```
{
```

```
orxVECTOR vSoldierPos;
```

```
orxObject_GetWorldPosition(pstSoldier, &vSoldierPos);
vPos.fZ = vSoldierPos.fZ;
```

```
orxObject_SetPosition(pstSoldier, &vPos);
```

}

{

在操作视口之前,我们先关注下视口所关联的摄像机,我们可以移动,旋转和缩放它。获取摄像机的代码如下所示:

```
pstCamera = orxViewport_GetCamera(pstViewport);
```

非常简单。让我们实现旋转。²⁾.

```
if(orxInput_IsActive("CameraRotateLeft"))
```

```
orxCamera_SetRotation(pstCamera, orxCamera_GetRotation(pstCamera) +
orx2F(-4.0f) * _pstClockInfo->fDT);
```

我们再次看到旋转的角度时间并不依赖于FPS而是时钟的DT_]我们也可以通过设置System这个配置选项来 设置旋转速度,而不是使用硬编码。

实现缩放如下:

```
if(orxInput_IsActive("CameraZoomIn"))
{
    orxCamera_SetZoom(pstCamera, orxCamera_GetZoom(pstCamera) * orx2F(1.02f));
}
```

因为这个代码没有使用时钟信息,所以他将会被时钟频率和帧率所影响。 最后让我们移动摄像机。

orxCamera_GetPosition(pstCamera, &vPos);

if(orxInput_IsActive("CameraRight"))

vPos.fX += orx2F(500) * _pstClockInfo->fDT;

}

{

```
orxCamera_SetPosition(pstCamera, &vPos);
```

好了,与摄像机有关的先到这里吧。在下面的配置中我们将看到,同一个摄像机被连接到两个不同的视口。操作摄像机将同时影响两个视口。

```
我们可以直接修改视口的位置和尺寸,如下所示:
orxFLOAT fWidth, fHeight, fX, fY;
orxViewport_GetRelativeSize(pstViewport, &fWidth, &fHeight);
if(orxInput_IsActive("ViewportScaleUp"))
{
  fWidth *= orx2F(1.02f);
  fHeight*= orx2F(1.02f);
}
orxViewport_SetRelativeSize(pstViewport, fWidth, fHeight);
orxViewport_GetPosition(pstViewport, &fX, &fY);
if(orxInput_IsActive("ViewportRight"))
{
  fX += orx2F(500) * _pstClockInfo->fDT;
}
```

```
orxViewport_SetPosition(pstViewport, fX, fY);
```

如上 所示,没有什么惊奇的,非常简单。

```
让我们来接着看看 viewport的配置方面的东西。
```

```
[Viewport1]
                  = Cameral
Camera
              = (0.5, 0.5, 0.0)
RelativeSize
RelativePosition = top left
BackgroundColor = (0, 100, 0) \sim (0, 255, 0)
[Viewport2]
Camera
                 = Camera2
RelativeSize
                = @Viewport1
RelativePosition = top right
BackgroundColor = (100, 0, 0) \sim (255, 0, 0)
[Viewport3]
                  = Camera3
Camera
RelativeSize
                = @Viewport1
RelativePosition = bottom left
BackgroundColor = (0, 0, 100) \sim (0, 0, 255)
[Viewport4]
                  = @Viewport1
Camera
RelativeSize
                 = @Viewport1
RelativePosition = bottom right
BackgroundColor = (255, 255, 0)#(0, 255, 255)#(255, 0, 255)
```

Last update: 2018/02/14 00:46 (7 years ago) cn:orx:tutorials:viewport https://wiki.orx-project.org/cn/orx/tutorials/viewport?rev=1518583597

我们可以看到,还是没有什么新的让人惊喜的东西。 一共有3个摄像机,它们关联了4个视口,其中Camera1关联了Viewport1和Viewport4[] 我们注意到Viewport1的配置文件中relativeSize设置为(0.5,0.5,0).它代表的意思在x轴和y轴方向上分 别使用一半的显示尺寸[]z轴被忽略)。也就是说,任何一个视口实际上显示部分的内容是可调的,可以是 全屏或者非全屏。

接下来我们注意到其他视口的RelativeSize属性被设置成@Viewport1[]它的意思是RelativeSize属性继承Viewport1的RelativeSize属性,也就是说它们的RelativeSize属性和Viewport1的RelativeSize属性一样。 我们也可以看到Viewport4的Camera属性被设置成@Viewport1,表明它继承自Viewport1的摄像机。

为了避免视口在屏幕中互相重叠遮盖,我们可以设置RelativePosition属性为常量字符³⁾或者使用vector设置它们的合理位置。

最后前三个视口使用随机的红色作为背景颜色,设置如下:

BackgroundColor = $(200, 0, 0) \sim (255, 0, 0)$

意思是这个viewpor将使用一个随机的红色.⁴⁾如果我们希望通过准确的随机颜色进行设置,可以使用一下列表的形式设置,随机的颜色分别为黄、青和品红,设置如下:

BackgroundColor = (255, 255, 0) # (0, 255, 255) # (255, 0, 255)

This gives three possibilities for our random color: yellow, cyan and magenta. 这种使用方式是相当于在 三个颜色(黄色, 蓝绿色, 品红)中进行随机。

最后让我们关注摄像机的设置。

[Cameral]					
FrustumWidth	=	@Displ	.ay.Sc	reenWidth	
FrustumHeight	=	@Displ	.ay.Sc	reenHeight	
FrustumFar	=	1.0			
FrustumNear	=	0.0			
Position	=	(0.0,	0.0,	-1.0)	
[Camera2]					
FrustumWidth	=	400.0			
FrustumHeight	=	300.0			
FrustumFar	=	1.0			
FrustumNear	=	0.0			
Position	=	(0.0,	0.0,	-1.0)	

[Camera3@Camera1]

我们仅仅定义了他们的 frustum(视锥)(被摄像机所拍摄的世界空间的一部分,将被映射到视口显示). NB:因为我们使用的"2D"的摄像头,视锥的形状是 frectangular cuboid(长方体).

我们可以发现Camera3完全继承自Camera1]它没有覆盖Camera1的任何属性。他们有完全一样的属性。 NB:使用完全继承所有属性可以写成[[MySection@ParentSection]] 为什么实用两个不同的提供处呢? 仅仅用为可以有两个不同的物理实体(phycical optitios)[]我们在代码中

为什么实用两个不同的摄像头呢? 仅仅因为可以有两个不同的物理实体(physical entities)[]我们在代码中修改了Camera1的属性,而 Camara3将保持不变。

我们注意到Camera1的FrustumWidth和FrustumHeight属性继承自Display的屏幕设置。

NB: 当继承某个属性,可以写成MyKey = @ParentSection.ParentKey.当两个key一样时,其中父选关键字可以省略如[]SameKey = @ParentSection.

最后我们注意到Camera2具有较小的视锥。

也就是说Camera2只能看到世界空间的较小部分。所以视口看起来具有了放大的效果!

资源

源代码: 05_Viewport.c

配置文件: 05_Viewport.ini

1)

比如,在HUD[]游戏运行时的状态栏)和UI[]界面)中很有用

其他方向仅仅只有部分代码,但是逻辑是一样的

由关键字 top,bottom,center,right和left组成

,~, 字符被用在两个数字之间, 作为随机操作符

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

Permanent link: https://wiki.orx-project.org/cn/orx/tutorials/viewport?rev=1518583597

Last update: 2018/02/14 00:46 (7 years ago)





viewport