Scrolling tutorial

Summary

See previous basic tutorials for more info about basic object creation, clock handling, frames hierarchy, animations, cameras & viewports, sounds & musics, FXs and physics.

This tutorial shows how to display a parallax scrolling.

As you can see, there's no special code for the parallax scrolling itself.
Actually, orx's default 2D render plugin will take care of this for you, depending on how you set the objects' attributes in the config file.

By default, in this tutorial, the attribute `AutoScroll` is set to 'both'.
This means a parallax scrolling will happen on both X and Y axis when the camera moves.
You can try to set this value to x, y or even to remove it.

Along the `AutoScroll` attribute, you can find the `DepthScale` one.
This attribute is used to automatically adjust the objects' scale depending on how far they are from the camera.
The smaller the camera frustum is, the faster this autoscale will apply.
You can try to play with object positionning and camera near & far planes to achieve the desired scrolling and depth scale rates you want.

You can change the scrolling speed (ie. the camera move speed) in the config file. As usual, you can modify its value in real time and ask for a config history reload.

As you can see, our update code simply moves the camera in the 3D space.
Pressing arrows will move it along X and Y axis, and pressing control & alt keys will move it along the Z one.
As told before, all the parallax scrolling will happen because objects have been flagged appropriately.
Your code merely needs to move your camera in your scenery, without having to bother about any scrolling effect.
This gives you a full control about how many scrolling planes you want, and which objects should be affected by it.

The last point concerns the sky.
As seen in the frame tutorial, we set the sky object's frame as a child of the camera.
This means the position set for the sky object in the config file will always be relative to the camera one.
In other words, the sky will always follow the camera.
As we put it, by default, at a depth of 1000 (ie. the same value as the camera far frustum plane), it'll stay in the background.

Details

As usual, we begin by creating a viewport, 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. 😀

Resources

Source code: 09_Scrolling.c

Config file: 09_Scrolling.ini