Object tutorial

Summary

As orx is data driven, here we only need two lines of code to create a viewport and an object. All their properties are defined in the config file (01_Object.ini).

The viewport is associated with a camera which is implicitly created from the info given in the config file. Still in this config file, you can also set their sizes and positions or the object color, scale, rotation, animation, physical properties, and so on. You can even request random values for all these setting without having to add a single line of code.

In a later tutorial we'll see how to generate complex object hierarchies or even a whole scene (all background and landscape objects for example) with only a single line of code.

For now, you can try to uncomment some of the lines of 01_Object.ini, play with them, then relaunch this tutorial. For an exhaustive list of options, please have a look at CreationTemplate.ini.

Details

Creating an object is really simple. However, we first need to make sure that we have loaded the config file where all the object's properties are defined. We also want to display the created object, through a viewport/camera set.

Don't panic! All this is really easy. 

In this tutorial, a config file which is in our parent directory is getting loaded. As you may see, in this case, all the executables are in their own child directory, depending on their build type (mingw, vs2005, vs2008, etc...) and we didn't want to duplicate the corresponding config files everywhere. [1)]

We then create our viewport. Note that the camera creation is automatically done given the config information stored for this viewport.

```
orxViewport_CreateFromConfig("Viewport");
```

We're almost done. We only need to create our object now!

```
orxObject_CreateFromConfig("Object");
```

That's it! The object has been created and will be displayed as long as it is in the frustum of our camera.

Now, we need to declare our “entry” points (here our `Init`, Run and `Exit` functions).

We then need run orx and let it know about those functions as well as the command line parameters. All this is done with a single line of code:

```c
int main(int argc, char **argv)
{
  orx_Execute(argc, argv, Init, Run, Exit);

  return EXIT_SUCCESS;
}
```

Calling `orx_Execute()` will also make sure the clock module is constantly ticked (as it's part of orx's core) and that we exit if the orxSYSTEM_EVENT_CLOSE event is sent.
This event is sent when closing the windows, for example, but it can also be sent under your own criteria (escape key pressed, for example).[2)]

As orx is data driven, we don't need to manually load any data, such as a sprites. Everything is handled for us by a data manager that will make sure sprites won't be duplicated in memory and freed when not used anymore.

If you look at the config file, in the [Object] section, you'll see that you can specify all the object's properties, such as: graphic (sprite), pivot, color, alpha, physics properties, position, rotation, scale, tiling (repeat), animation, visual FX, shader, etc...

Don't worry, all this will be covered in further tutorials.

Now we have an object, we need to learn how to interact with it. This brings us to our second tutorial: clock.

Resources

Source code: 01_Object.c

Config file: 01_Object.ini

1)
However, if your config file name matches your executable and is in the same folder, it'll be loaded automatically.
2)
more details on this and the whole game loop in the C++/localization tutorial