Render to External Window

By using a native plugin you are able to render a camera's view to an external window. The package provides plugins for both Windows (using Direct3d 11) and MacOS (using Metal).

Once you're importing this package, open the Demo.unity scene from /Assets/RenderToOtherWindow/Scenes.

In the demo scene, you'll notice two cameras, "Main Camera" and "MovingCamera". The second camera's target will be a render texture(created at runtime) which will be accessed by the external window to display what the moving camera (in this demo scene) sees.

Inside the /Assets/RenderToOtherWindow/Scripts folder you'll notice some scripts which have a meaning only in the context of this demo scene.
MovingCameraScript.cs is just for moving the second camera, so, to make a better distinction compared to what the Main Camera is rendering.
ButtonBehaviour.cs is for displaying/closing the external window when the user clicks the button. It actually calls the ***PluginInterface.cs***, which is explained in this document.
BehaviourScript.cs is just for placing the human on sofa.

center@republicofhandball.com

PluginInterface.cs

This script is a wrapper for calling the native plugin **ExternalWindowPlugin** , which can be found in /Assets/RenderToOtherWindow/Plugins. The native plugin to be used has .**dll** or .**bundle** extension, for Windows or MacOS respectively.

The script has only one public method: **ToggleShowWindow**(). Which has to be called for showing/closing the external window.

Below are presented the methods exported by the native plugin. *SetWindowRect* is not called in this example, but is very likely to be needed, in some scenarios.

void StartWindow(string title, int width, int height, bool borderless = false) //show the window with the given size, and title; If the borderless is true, there will //be no title bar for the window.

void StopWindow() //stop the plugin, destroy the window and release any //graphic resources (Direct3D or Metal).

void SetWindowRect(int left, int top, int width, int height) //change the //window's size and position

void SendTextureIdToPlugin(IntPtr texId); //sends the texture's pointer //(Direct3D case) or the texture's id (Metal case), to the native plugin

IntPtr InitGraphics(); //initialize the graphics; it gets called //on the low level rendering thread
IntPtr GetRenderEventFunc(); //called on each frame, on the low level //rendering thread, asking the graphics to render and present the data in the //external window

In PluginInterface.cs, inside StartPlugin(), it can be noticed, that the render texture is created just before asking the native plugin to create the window. The render texture and the window has the same size, but, obviously, if someone wants, he could use different sizes. If the user resizes the window, after it has been created, either manually or by calling SetWindowRect, the render texture is not recreated.

Newly added(for Windows):

It is possible to set the transparency for the external window, either by the color key or by alpha level, or both, check the scene /Scenes/DemoTransparency.unity:

```
static extern void SetWindowAttributesById(int windowId, int color, byte alpha, int flags);
flags:
-  0 : no layering(both color and alpha are ignored)
```

- 1 : color key transparency (each pixel with exactly that value is fully transparent; alpha value is ignored)
- 2 : alpha transparency (whole window is transparent with the given alpha level)
- 3 : both color key transparency and alpha transparency

It is also possible to have more external windows, check please the scene /Scenes/DemoMultipleWindows.unity. As you can see, the names of the methods have been suffixed with "ById" and a new parameter has been added "windowId" to identify the respective external window. The previous methods have been kept as for Mac OS this feature is yet to be added.

```
    static extern void StartWindowById(int windowId, [MarshalAs(UnmanagedType.LPWStr)]
string title, int width, int height, bool borderless = false);
    static extern IntPtr StopWindowById(int windowId);
    static extern void SetWindowRectById(int windowId, int left, int top, int width, int
height);
    static extern void SendTextureIdToPluginById(int windowId, IntPtr texId);
```

To select a different texture format for Windows plugin, like `ARGBHalf` instead of `ARGB32`, modify line 153 in PluginInterface.cs. The formats `ARGB32, ARGBHalf` can be selected, if other formats are desired, a similar change in the plugin code has to be done(and so the plugin has to be recompiled). In main.cpp `DXGI_FORMAT_R8G8B8A8_UNORM`, `DXGI_FORMAT_R16G16B16A16_FLOAT` can be selected, so in SetColorFormat method additional formats could be added.