**Oculus PC SDK Engine Integration**

This document is a first stop for any developer with an existing PC game engine, who wishes to incorporate the Oculus SDK.

You’ll need to download the Oculus Runtime, from [www.oculus.com](http://www.oculus.com) which you’ll then need to install. Also, download and unzip the Oculus SDK. Now, to check everything’s working, open up the “Oculus Room Tiny” demo (ORT), and check it runs. Most of the components of this integration description can be lifted straight out of the main file of that demo, so its good to have it available and working.

**As a companion to this document, look at the sample “Engine Integration Stages” to see a compilable, and executable, sample app going through the stages to integrate itself and underlying engine, with the Oculus SDK.**

**Stage 1 – Start with a nonVR application**

As a starting point, we’ll assume you have a simple non-VR test application running in your engine, displaying on your monitor. We’ll then talk you through integrating the components of VR.

**Stage 2 – Add LibOVR**

Include the Oculus library, with its CAPI interface, and initialise the SDK and your Rift. This should be done as one of the first things in your engine, certainly before initialising the window, as the window will need details from the library. Similarly, add the release functions in your engine release, after other items, ideally as the last thing called. Check that no errors are flagged at this stage.

**Stage 3 – Render to eye buffers**

Instead of rendering to the main screen, you now must readjust your engine to render to two separate eye buffers. These will ultimately be passed to the SDK for distortion and correct display, but for now, allocate them according to the SDK parameters and the resolution you wish, and direct the output of your game to both of these in turn. Render these textures to the screen to ensure everything is as it should be.

**Stage 4 – Hook into the sensors**

Note we are still rendering to a conventional monitor throughout these early stages, as this makes debug easier. Now interrogate the Rift sensors, and feed these into the cameras used to render your two eye buffers. You should thus be able to check that movements of the Rift correctly affect your view, both orientation changes, and positional changes. In particular, note that your two eye cameras are positionally apart, reflecting the fact that your eyes are apart, and thus that everything will be perceived stereoscopically.

**Stage 5 – Output to the headset**

Now we're going to put it on the output onto the Rift, and for this we’ll be modifying the window location and size, and also register it in with Direct Mode with the 'attach window' function. Note, it won't look right on the Rift yet, as we haven't located and distorted the eye buffers correctly yet. It’s advisable to use Direct mode

in order to see clearly what is happening, but make sure that both modes are set up to output to the headset.

**Stage 6 – Add distortion**

Finally, we allow the SDK to render and distort the eye buffers, including present.

Note the addition of a timing call at the start of the frame, and you may wish to provide a mechanism to dismiss the Health and Safety warning, perhaps on a key.

**Further checks on engine integration**

* Its recommended that you implement a timewarp freeze function, as demonstrated in the samples (“OculusRoomTiny (Freeze Timewarp)”), to ensure your timewarp is set up correctly, especially if you have a left-handed-coordinate system, and have had to swap and negate values along the way.
* It is recommended you implement an IPD varying debug parameter (see sample “OculusRoomTiny(Adjust IPD)”) to ensure that you have not inadvertently created a monoscopic application.
* Ensure that you framerate is consistent and correct, e.g. 75Hz on DK2.
* This document describes a basic setup – for adjustments and further techniques, please refer to the extensive Oculus Room Tiny samples as part of the SDK.