DXR Tutorial 1

Creating a Window

# Introduction

Hello everyone and welcome to the DXR tutorial series. Over the course of the next 14 tutorials, we will cover all the basics of the DXR API.

The tutorials will only cover the DXR API. It is assumed that the reader has good knowledge of DirectX12 and Windows programming.

Let’s begin…

# Falcor

One of the first things you’ll notice, is that the tutorials are packaged together with Falcor. Falcor is a real-time rendering framework which supports many graphics features and abstracts the graphics API. We will not use any of those features in the tutorials, except for the window abstractions. Like I mentioned before – it is assumed that the user knows how to create a window and handle Windows messages. Everything else besides the window will be implemented directly by us, using raw-API calls.

# In this tutorial we will not actually do anything specific to DXR, but rather have a quick overview of how to write a Falcor sample.

# 01-CreateWindow.h

This is where most of the code for the tutorial is.

We start by including **“Falcor.h”**, which contains some required declarations for the window-abstraction. If you drill down into the included file, you’ll get to **“FalcorD3d12.h”**. This if where things differ from a regular D3D12 application – as you can see we include a new header file - **“d3d12\_1.h”**. This header contains all the regular D3D12 declarations, as well as the new DXR declarations.

Let’s get back to 01-CreateWindow.h.

The only class in this file is **DxrSample** which inherits from **Renderer**. This base class provides the windows and messaging abstraction. As you can see, we are overriding 5 base-class functions. These functions are callbacks which will be called during different times in the application’s lifetime:

**onLoad()**

Called once at the beginning. The window is already created at this point and we can use the window handle safely. This is where we will initialize the different API objects.

**onFrameRender()**

The main render function. This is where will create and submit graphics commands.

**onShutdown()**

Called right before the application terminates. This is where we want to place all the cleanup code. You’ll notice that we are using smart pointers and smart COM pointers, so this function is not really used.

**onKeyEvent() / onMouseEvent()**

Called whenever we get a mouse or keyboard event from Windows. Not used in these tutorials.

We can ignore the input parameters to those functions – they will not be used in the tutorials.

**Renderer** provides a bunch of other callbacks, but they are not used in these tutorials and will not be covered.

Well, that’s it for the header. Time to move to the CPP file.

# 01-CreateWindow.cpp

Nothing much here. We included **“01-CreateWindow.h”** and added empty definitions of the required callback.

If you’ll scroll to the bottom of the file, you’ll see the ***WinMain()***function. It doesn’t do much.

First, we instantiate a **DxrSample** object using a unique\_ptr.

Then, we create and initialize a **SampleConfig** object. It has many fields, but we only care about 2 of them.

* config.windowDesc.title **=** "Tutorial 01 - Create Window"**;**

This line sets the window title.

* config.flags = SampleConfig::Flags::DoNotCreateDevice;

Here we set a flag that tells Falcor not to create a device. We will create the device ourselves in the next tutorial.

Once we initialize the configuration, we are ready to go. Calling Sample::run(config, pSample) will start Falcor’s render loop, which will call our callbacks when suitable.

Running this application doesn’t yield much. Just an empty window.

# Conclusion

We didn’t do much here. Just creating a window. In the next tutorial we will start using DXR.