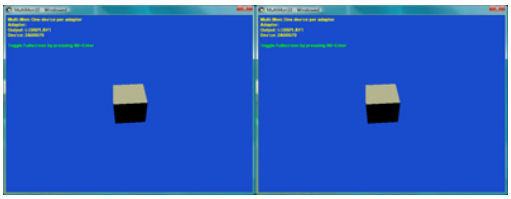
MultiMon10 Sample

Collapse All

This sample demonstrates how to use multiple monitors with Direct3D 10.



Path

Source	*SDK root* \Samples\C++\Direct3D10\MultiMon10
Executable	*SDK root* \Samples\C++\Direct3D10\Bin\ *platform* \MultiMon10.exe

How the Sample Works

The sample works by enumerating all Direct3D 10 adapters and outputs on the system. In Direct3D 10 an adapter refers to the physical display hardware in the machine. An output refers to an output device connected to that physical display hardware. The output device is most often a monitor. Many display adapters will have more than one output attached. For example, if your video card has two DVI connectors on the back, Direct3D 10 will most likely enumerate this as on adapter with two outputs attached.

To render to multiple monitors, the sample creates one Direct3D 10 device per adapter, and one swapchain per output. On a system like the one described above, this would be one Direct3D 10 device with two swapchains attached to it.

Enumerating Adapters and Outputs

The **EnumerateAdapters** functions enumerates both the display adapters in the machine and the outputs connected to those display adapters. **IDXGIFactory::EnumAdapters** functions in much the same as the GDI enumeration functions. You keep passing in a higher and higher adapter number into the function until it returns DXGI_ERROR_NOT_FOUND, which is the clue that there are no more adapters. Each adapter has number of outputs that can be attached to it. These too, must be enumerated.

Setting Up MultiMon

Once the adapters and outputs have been enumerated, the sample setups a typical multi-monitor scenario. Each monitor will have one window on it. Each adapter will have one Direct3D 10 device. Each output attached to that adapter will have one swapchain. It is important to note that unless sharing is used, resources created on one device cannot be accessed from another device. To illustrate this, each device that is created will create and draw a different shape.

Rendering to Multiple Monitors

Once a swapchain has been associated with each window/output, rendering can begin. **RenderToAllMonitors** renders to each window/output sequentially. In order to render to a specific output, the render target and viewport must be set to those specific to that swapchain.

```
// set the render target
pWindowObj->pDeviceObj->pDevice->OMSetRenderTargets( 1, &pWindowObj->pRenderTargetView, pWindowObj->pDepthStencilView );
// set the viewport
D3D10_VIEWPORT Viewport;
Viewport.TopLeftX = 0;
Viewport.TopLeftY = 0;
Viewport.Width = pWindowObj->Width;
Viewport.Height = pWindowObj->Height;
Viewport.MinDepth = 0.0f;
Viewport.MaxDepth = 1.0f;
pWindowObj->pDeviceObj->pDevice->RSSetViewports( 1, &Viewport );
```

After this, the window specific render function can be called. **PresentToAllMonitors** presents to each swapchain associated with a window to get the final image on the screen.

Toggling Fullscreen

This sample uses DXGI's Alt+Enter handling to toggle between full-screen and windowed mode. When DXGI handles Alt+Enter, the application only needs to handle WM_SIZE messages to realize that the window size has changed, and therefore, the backbuffer needs to be resized. When a WM_SIZE message is received, the sample first finds the WINDOW_OBJECT that corresponds to the window handle that received the message. Once this is found, the render target view, depth stencil view, and depth stencil surface are released. The swapchain is resized based upon the dimensions of the window's client rect. Finally, the render target view, depth stencil surface, and depth stencil view are recreated to match the new swapchain.

```
DXGI_SWAP_CHAIN_DESC Desc;
pObj->pSwapChain->GetDesc( &Desc );

pObj->pSwapChain->ResizeBuffers( Desc.BufferCount,
                                 (UINT)rcCurrentClient.right,
                                 (UINT)rcCurrentClient.bottom,
                                 Desc.BufferDesc.Format,
                                 0 );

pObj->Width = (UINT)rcCurrentClient.right;
pObj->Height = (UINT)rcCurrentClient.bottom;

// recreate the views
CreateViewsForWindowObject( pObj );
```