

Teaching Myself DirectX

The Device

A device is used to create resources and to enumerate the capabilities of a display adapter. A Direct3D device allocates and destroys objects, **renders primitives**, and **communicates with a graphics driver and the hardware**. In Direct3D 11, a device is separated into a device object for creating resources and a device-context object, which performs rendering. A device is represented with an [**ID3D11Device**](https://docs.microsoft.com/en-us/windows/desktop/api/D3D11/nn-d3d11-id3d11device) interface.

**Each application must have at least one device.**

[**D3D11CreateDevice**](https://docs.microsoft.com/en-us/windows/desktop/api/D3D11/nf-d3d11-d3d11createdevice) or [**D3D11CreateDeviceAndSwapChain**](https://docs.microsoft.com/en-us/windows/desktop/api/D3D11/nf-d3d11-d3d11createdeviceandswapchain) can be used to create a device.

Device Context

A device context is used to set **pipeline** state and generate rendering commands using the resources owned by a device.

Swap Chain

A swap chain is a collection of buffers that are used for displaying frames to the user. Each time an application presents a new frame for display, the first buffer in the swap chain takes the place of the displayed buffer. This process is called swapping or flipping.

A graphics adapter holds a **pointer** to a surface that represents the image being displayed on the monitor, called a **front buffer**. As the monitor is refreshed, the graphics card sends the contents of the front buffer to the monitor to be displayed. However, this leads to a problem when rendering real-time graphics. The heart of the problem is that monitor refresh rates are very slow in comparison to the rest of the computer. Common refresh rates range from 60 Hz (60 times per second) to 100 Hz. If your application is updating the front buffer while the monitor is in the middle of a refresh, the image that is displayed will be cut in half with the upper half of the display containing the old image and the lower half containing the new image. **This problem is referred to as tearing**.

Back buffering is the process of drawing a scene to an off-screen surface, called a back buffer. Note that any surface other than the front buffer is called an **off-screen surface because it is never directly viewed by the monitor**. By using a back buffer, an application has the freedom to render a scene whenever the system is idle (that is, no windows messages are waiting) without having to consider the monitor's refresh rate. Back buffering brings in an additional complication of how and when to move the back buffer to the front buffer.

The process of moving the back buffer to the front buffer is called surface flipping. Because the graphics card simply uses a pointer to a surface to represent the front buffer, a simple pointer change is all that is needed to set the back buffer to the front buffer. When an application asks Direct3D to present the back buffer to the front buffer, Direct3D simply "flips" the two surface pointers. The result is that the back buffer is now the new front buffer, and the old front buffer is the new back buffer. A surface flip is invoked whenever an application asks the Direct3D device to present the back buffer; however, Direct3D can be set up to queue the requests until a vertical sync occurs. This option is referred to as the Direct3D device's presentation interval. Note that the data in the new back buffer may not be reusable, depending on how an application specifies how Direct3D should handle surface flipping. Surface flipping is key in multimedia, animation, and game software; it is equivalent to the way you can do animation with a pad of paper. On each page, the artist changes the figures slightly, so that when you flip rapidly between sheets, the drawing appears animated.

Creating Device , Context and Swap Chain at once

HRESULT D3D11CreateDeviceAndSwapChain(

IDXGIAdapter \*pAdapter,

D3D\_DRIVER\_TYPE DriverType,

HMODULE Software,

UINT Flags,

const D3D\_FEATURE\_LEVEL \*pFeatureLevels,

UINT FeatureLevels,

UINT SDKVersion,

const DXGI\_SWAP\_CHAIN\_DESC \*pSwapChainDesc,

IDXGISwapChain \*\*ppSwapChain,

ID3D11Device \*\*ppDevice,

D3D\_FEATURE\_LEVEL \*pFeatureLevel,

ID3D11DeviceContext \*\*ppImmediateContext

);

1. A pointer to the video adapter to use when creating a [device](https://docs.microsoft.com/en-us/windows/desktop/direct3d11/overviews-direct3d-11-devices-intro). Pass **NULL** to use the default adapter, which is the first adapter enumerated by [IDXGIFactory1::EnumAdapters](https://docs.microsoft.com/en-us/windows/desktop/api/dxgi/nf-dxgi-idxgifactory-enumadapters).
2. The [D3D\_DRIVER\_TYPE](https://docs.microsoft.com/en-us/windows/desktop/api/d3dcommon/ne-d3dcommon-d3d_driver_type), which represents the driver type to create.

typedef enum D3D\_DRIVER\_TYPE {

D3D\_DRIVER\_TYPE\_UNKNOWN = 0,

D3D\_DRIVER\_TYPE\_HARDWARE,

D3D\_DRIVER\_TYPE\_REFERENCE,

D3D\_DRIVER\_TYPE\_NULL,

D3D\_DRIVER\_TYPE\_SOFTWARE,

D3D\_DRIVER\_TYPE\_WARP

} ;

<https://docs.microsoft.com/en-us/windows/win32/api/d3dcommon/ne-d3dcommon-d3d_driver_type>

<https://docs.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-d3d11createdeviceandswapchain>

***\*\*This function takes device and swap-chain descriptions and creates device , swap-chain and device context .***

Direct3D interface pointers(Windows interface pointers) need to be called release function for realising the object(freeing the memory) ,for that we should use ComPtr (under wrl.h header)

**Microsoft::WRL::ComPtr<TypeName> ptr**

Microsoft::WRL::ComPtr<ID3D11Device> pDevice;

Microsoft::WRL::ComPtr<IDXGISwapChain> pSwapChain;

Microsoft::WRL::ComPtr<ID3D11DeviceContext> pDeviceContext;

Microsoft::WRL::ComPtr<ID3D11Resource> pBackBuffer;

Microsoft::WRL::ComPtr<ID3D11RenderTargetView> pTarget;

**ID3D11 vs IDXGI**

IDXGI is those interfaces which doesn’t change as frequently as direct3D ( Direct3D9 , Direct3D10, Direct3D11 , Direct3D12)

*It basically factored all the enumeration, display and adapter management, and presentation stuff out of Direct3D. That way, all sorts of graphics APIs can coexist without a need to have separate mechanisms for these common tasks in each of them. It allows, e.g., all the Direct3D APIs (>= 10) to only be concerned with drawing 3D content into buffers and not care about where these buffers come from, or whether and how they are going to be displayed.*