BasicCompute11

<https://github.com/walbourn/directx-sdk-samples>

This is the DirectX SDK's Direct3D 11 sample updated to use Visual Studio 2012 and the Windows SDK 8.0 without any dependencies on legacy DirectX SDK content. This sample is a Win32 desktop DirectX 11.0 application for Windows 10, Windows 8.1, Windows 8, Windows 7, and Windows Vista Service Pack 2 with the DirectX 11.0 runtime.

**This is based on the legacy DirectX SDK (June 2010) Win32 desktop sample. This is not intended for use with Windows Store apps, Windows RT, or universal Windows apps.**

# Description

This sample shows the basic usage of DirectX11 Compute Shader (aka DirectCompute) by implementing array A + array B.

https://code.msdn.microsoft.com/site/view/file/94433/1/BasicCompute11.jpg

## How the Sample Works

Setting up the Compute Shader involves the following steps:

1. Create a D3D11 device and context. Make sure to check the feature level of the device created. Based on what graphics card is installed in the system, possibilities are:
   1. If an FL11 device has been created, we get full Compute Shader 5.0 capability.
   2. However, if we have an FL10 or FL10.1 device, Compute Shader 4.0/4.1 is potentially available, since CS4.0/4.1 is available on most DirectX 10 cards but not all of them. Call CheckFeatureSupport to see whether CS4.0/4.1 is available. Refer to the sample code to see how this is done.
   3. If we get an FL9.x device, Compute Shader is not available.
2. Compile and then create the Compute Shader.
3. Create input resource for the Compute Shader and fill them with data. As we are doing array A + array B, we create two buffers as the input resource.
4. Create an SRV (shader resource view) for both of the input buffer resources. Shader resource view is used to bind input resources to shaders.
5. Create output resource for the Compute Shader.
6. Create a UAV (unordered resource view) for the output resource. Unordered resource view is used to bind output resources to Compute Shaders. CS4.0/4.1 can have only one output resource bound to a Compute Shader at a time. CS5.0 doesn't have this limitation.
7. Execute the Compute Shader by calling Dispatch.

## Build Options

The sample and the HLSL code supports two additional build modes controlled by compile-time defines.

* USE\_STRUCTURED\_BUFFERS: If this is defined, then the sample and shader uses structured buffer types. If this is commented out, then the sample and shader use raw buffer types.
* TEST\_DOUBLE: If this is defined, then the sample tests a double-precision data type. Note that double support requires Compute Shader 5.0 and optional driver support for double-precision shaders. If the sample and shader are compiled with TEST\_DOUBLE defined, then the sample will only run on 11-class hardware with doubles support, or it will fall back to using the Reference device.

# Dependencies

Samples typically make use of runtime HLSL compilation. Build-time compilation is recommended for all production Direct3D applications, but for experimentation and samples development runtime HLSL compilation is preferred. Therefore, the D3DCompile\*.DLL must be available in the search path when these programs are executed.

* When using the Windows 8.x SDK and targeting Windows Vista or later, you can include the D3DCompile\_46 or D3DCompile\_47 DLL side-by-side with your application copying the file from the REDIST folder.

%ProgramFiles(x86)%\Windows kits\8.0\Redist\D3D\arm, x86 or x64

%ProgramFiles(x86)%\Windows kits\8.1\Redist\D3D\arm, x86 or x64

%ProgramFiles(x86)%\Windows kits\10\Redist\D3D\arm, x86 or x64

# More Information

[Where is the DirectX SDK (2015 Edition)?](https://walbourn.github.io/where-is-the-directx-sdk-2015-edition/)

[Games for Windows and DirectX SDK blog](https://walbourn.github.io/)