NVIDIA Capture SDK

Sample Description Document

2016-2018

Version 7.0

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Module Documentation

2.1 GLIFRPerfHwEnc

This sample is a performance test for OpenGL NVIFR capture and H.264 hardware encoding. Settings used are 2-pass quality with low latency HP at 720p resolution, 5 mbps, and 30 fps.

Performance test for OpenGL NvIFR capture and H264 encoding, with following parameters - Rate control mode: 2_PASS_QUALITY Preset : LOW_LATENCY_HP Resolution : 720p FramesPerSecond : 30 Bitrate : 5000 $*$ 1000

2.2 PerfNVHWEnc

This sample acts as a benchmark to measure the maximum performance of the NVIFR encoder.

2.3 DX10-AsyncHWEncode

This sample shows how to use the NVIFR INVIFRToHWEncoder Interface with DirectX 10 to capture a render target, compress it to H.264 or HEVC, and write it to a video file. It demonstrates using asynchronous threads to perform rendering, encoding, and write to file.

2.4 DX10-SimpleSample

This sample shows how to use NvIFR with DirectX 10 and how to capture a render target to a file.

It shows how to load the NvIFR dll, initialize the function pointers, and create the NvIFRToSys object.

It then goes on to show how to set up the target buffers and events in the NvIFRToSys object and then calls TransferRenderTargetToSys to transfer the render target to system memory.

2.5 DX11-AsyncHWEncode

This sample shows how to use the NVIFR INVIFRToHWEncoder Interface with DirectX 11 to capture a render target, compress it to H.264 or HEVC, and write it to a video file. It demonstrates using asynchronous threads to perform rendering, encoding, and write to file.

2.6 DX11-SimpleSample

This sample shows how to use NvIFR with DirectX 11 and how to capture a render target to a file.

It shows how to load the NvIFR dll, initialize the function pointers, and create the NvIFRToSys object. It then goes on to show how to set up the target buffers and events in the NvIFRToSys object and then calls TransferRenderTarget-ToSys to transfer the render target to system memory.

2.7 DX9-AsyncHWEncode

This sample shows how to use the NVIFR INVIFRToHWEncoder Interface with DirectX 9 to capture a render target, compress it to H.264 or HEVC, and write it to a video file. It demonstrates using asynchronous threads to perform rendering, encoding, and write to file.

2.8 DX9IFRHWEncMultiGPUStress

This DirectX 9 sample demonstrates how to grab and encode a frame using a shared surface with asynchronous render, and encode DX9 devices running in separate threads. This example shows a method to use all of the available GPUs in a system. In addition, however, writing to a shared surface also allows rendering to occur asynchronously, so that both rendering DX9 device and encoding DX9 device can run at their maximum rate. This sample uses NvIFRHWEncoder interface, which can generate H.264 as well as HEVC output.

2.9 DX9IFRSharedSurfaceHWEncode

This DirectX 9 sample demonstrates how to grab and encode a frame using a shared surface with asynchronous render and encode using different DX9 devices for rendering and encoding.

Like the AsyncH264HwEncode sample, writing the output file occurs asynchronously from the main thread. However, writing to a shared surface also allows rendering to occur asynchronously from encoding, so that both render and grab run at their maximum rate.

2.10 DX9IFRSimpleHWEncode

This DirectX 9 sample demonstrates how to capture a render target, compress it, and write it to a video file in a simple single threaded application using the NVIFR INVIFRToHWEncoder Interface.

This DX9 sample demonstrates grabbing and encoding a frame asynchronously using the NvIFR interface, encoding it to H.264, and writing to a video file. Rather than blocking the render thread waiting for the grab + encode to finish, it continues rendering and signals another thread when the encoded frame is ready.

2.11 DX9-SimpleSample

This sample shows how to use NvIFR with DirectX 9 and how to capture a render target to a file.

It shows how to load the NvIFR dll, initialize the function pointers, and create the NvIFRToSys object.

It then goes on to show how to set up the target buffers and events in the NvIFRToSys object and then calls TransferRenderTargetToSys to transfer the render target to system memory.

2.12 DXIFR_Shim

This DirectX 9/9Ex/10.0/10.1/11.0 sample demonstrates how to implement a shim layer to intercept D3D API calls, and perform NVIFR encoding on the rendered frames. It employs a side D3D device dedicated for NVIFR encoding, and the rendered frames are copied by NVIFR extension (please refer to NVIDIA Capture SDK Programming Guide for details) for D3D9 and D3D surface sharing (please refer to MSDN for details) for D3D9Ex, D3D10 and above. This sample cant run as a standalone, but should run with a D3D application or game. The shim DLLs, instead of the original DLLs, must be loaded by the D3D application. You can achieve that by placing the shim DLLs into the application directory. The original DLL must be renamed by adding an underscore prefix. For example, d3d9.-dll should be renamed as _d3d9.dll. After running, a file named NvIFR.h264 will be generated which contains the encoded H264 stream.

2.13 Multihead

This sample demonstrates how to grab the frame buffers from multiple displays using NvFBC. This is done by creating multiple NvFBC objects (one for each adapter), incrementing the NVFBC_TARGET_ADAPTER environment variable before each NvFBC_CreateFunction call, up to the maximum number of adapters in the system. This is accomplished with a helper class in NvFBCLibrary.h (in the Util folder) that initializes the NvFBC dll, sets pointers to the dll functions, and provides methods to set the TargetAdapter and create the NvFBC device.

2.14 NvFBCCudaNvEnc

This sample demonstrates how to use the NvFBCToCuda interface to copy the desktop into a CUDA buffer. From the CUDA buffer, this is then mapped directly to NVENC, where the NVENC hardware video encoder can encode the stream. We recommend that developers use the NVIDIA Video Codec to utilize the NVENC hardware video Encoder. This sample provides a useful wrapper class around the NVENC API, in Encoder.h.

2.15 NvFBCCudaSimple

This sample demonstrates how to use the NvFBCToCuda interface to copy the desktop into a CUDA buffer. It covers loading the NvFBC.dll, loading the NVFBC function pointers, creating an instance of NvFBCCuda, and using NvFBCCuda to copy the frame buffer into a CUDA device pointer.

2.16 NvFBCCursorCapture

This sample demonstrates how to use NVFBCs cursor capture feature to copy the desktop and cursor to system memory, where the cursor and desktop are grabbed using separate threads.

2.17 NvFBCDX9ClassificationMap

This sample demonstrates how to use the NvFBCToDX9 to capture Classification maps, and also to capture the desktop to DX9 video memory surfaces. This sample also generates a downscaled copy of the frame buffer to a bitmap, and saves it to file.

Reader.h: Declares the Reader class, this class is simply a wrapper around the NVENC reader API.

Reader.cpp: Defines the Reader class, wraps up the details when using the NVENC video reader so it don't detract from the NvFBCDX9 example.

NvFBCDX9ClassificationMap.cpp: Demonstrates usage of the NvFBCToDX9 interface with the ClassificationMap capture API. It demonstrates how to load NvFBC.dll, initialize NvFBC function pointers, create an instance of NvFBCDX9 and using NvFBCDX9 to extract a ClassificationMap from the driver, and also capture a downscaled version of the desktop.

2.18 NvFBCDX9DiffMap

This sample demonstrates how to use the NvFBCToDX9 to capture diff maps, and also to capture the desktop to DX9 video memory surfaces. This sample also generates a downscaled copy of the frame buffer to a bitmap, and saves it to file.

Reader.h: Declares the Reader class, this class is simply a wrapper around the NVENC reader API.

Reader.cpp: Defines the Reader class, wraps up the details when using the NVENC video reader so it don't detract from the NvFBCDX9 example.

NvFBCDX9DiffMap.cpp: Demonstrates usage of the NvFBCToDX9 interface with the DiffMap capture API. It demonstrates how to load NvFBC.dll, initialize NvFBC function pointers, create an instance of NvFBCDX9 and using NvFBCDX9 to extract a diffmap from the driver, and also capture a downscaled version of the desktop.

2.19 NvFBCDX9NvEnc

This sample demonstrates how to use the NvFBCToDX9 to capture to a DirectX 9 surface, and then send it to the NVENC encoder. This sample also demonstrates how to enable NVFBC Image Area Classification feature and pair it seamlessly with NVENC Emphasis Level Map feature for tweaking visual quality of Capture+Encoded video stream This sample provides a useful wrapper class around the NVENC API, in Encoder.h.

From the DX9 Buffer, we use VideoProcessBlt to transferred to NVENC where the H.264 video encoder can encode the stream.

Encoder.h: Declares the Encoder class, this class is simply a wrapper around the NVENC encoder API.

Encoder.cpp: Defines the Encoder class, wraps up the details when using the NVENC video encoder so it don't detract from the NvFBCDX9 example.

NvFBCDX9NvEnc.cpp: Demonstrates usage of the NvFBCToDX9 interface. It demonstrates how to load NVFBC.dll, initialize NvFBC function pointers, create an instance of NVFBCToDX9Vid interface and using NVFBCToDX9Vid APIs to copy the frame buffer into a DX9 device pointer, which is then passed to NVENC encoder for video encoding.

2.20 NvFBCDx9NvEncSharedSurface

The sample demonstrates how to use NVFBC to grab the desktop to a DX9 shared surface, and then send it to the hardware encoder to encode as H.264 or HEVC using a separate DX9 context.

Encoder.h: Declares the Encoder class, this class is simply a wrapper around the NVENC encoder API.

Encoder.cpp: Defines the Encoder class, wraps up the details when using the NVENC video encoder so it don't distract from the NvFBCDX9 example.

NvFBCDX9NvEnc.cpp: Demonstrates usage of the NvFBCToDX9 interface. It demonstrates how to load vFBC.dll, initialize NvFBC function pointers, create an instance of NvFBCDX9 and using NvFBCDX9 to copy the frame buffer into a DX9 device pointer, where it was VideoProcessBlt, where is then passed to NVENC to generate a H.264 video.

2.21 NvFBCEnableAPI

This sample demonstrates how to use NvFBC_Enable API exported by NVFBC library to programmatically enable or disable NVFBC feature. The API needs to be called from an application that is running with administrator privileges.

2.22 NvFBCHWEncode

This sample demonstrates how to use the INVFBCToHWEncoder interface to grab the desktop and encode to H.264 or HEVC using NVENC HW encoder in a single API call. The INVFBCToHWEncoder interface will be deprecated after NVIDIA Capture SDK 5.0. The program show how to initialize the NvFBCHWEnc encoder class, set up the grab and encode, and grab the full-screen framebuffer, compress it to HWEnc, and copy it to system memory. Because of this, the NvFBCHWEnc class requires a Kepler video card to work. Attempting to create an instance of that class on earlier cards will result in the create call failing.

2.23 NvFBCToSys

Demonstrates the use of NvFBCToSys to copy the desktop to a system memory buffer and save it as a file. This sample demonstrates the use of NvFBCToSys class to record the entire desktop. It covers loading the NvFBC DLL, loading the NvFBC function pointers, creating an instance of NvFBCToSys and using it to copy the full screen frame buffer into system memory.

2.24 NvFBCToSysClassificationMap

Demonstrates the use of NvFBCToSys to copy the desktop to a system memory buffer and save it as a file. It covers loading the NvFBC DLL, loading the NvFBC function pointers, creating an instance of NvFBCToSys and using it to copy the full screen frame buffer into system memory. This sample also demonstrates enabling NVFBC Image Area Classification Map feature. Default stampsize selected for image classification feature is 16x16. The classified frame output is stored in a .txt file.

2.25 GLIFRAsync

This OpenGL sample shows how to asynchronously transfer an OpenGL FBO Rendertarget while rendering the next frame.

The framerate is printed to the window. Async transfers can be toggled by pressing 'a'.

2.26 GLIFRMultiHwEnc

This OpenGL sample shows how to use NvIFROGL to encode the contents of a FBO RenderTarget with the H.264 hardware encoder using multiple GPUs. One thread is created for each stream on a GPU.

2.27 GLIFRNVENCDynRes

This sample shows how to handle dynamic resolution changes while using NvIFROGL to encode the contents of a FBO frame buffer with the hardware H.264 encoder.

This sample shows how to use NvIFROGL to encode the contents of a framebuffer with dynamically increasing width/height of encoded image, increasing bit rate and both bit rate and resolution.

2.28 GLIFRNVENCGetCaps

This sample shows how to use NvIFROGL API to check for capabilities and supported features of the HW Encoder.

This sample shows how to use NvIFROGL to get encode caps of the hardware encoder for given codecType

2.29 GLIFRPbufferHwEnc

This sample shows how to use NvIFROGL to encode the contents of a pbuffer with the H.264 hardware encoder.

2.30 GLIFR_Shim

This sample shows how to use NvIFROGL to encode the contents of a FBO framebuffer with the hardware H.264 encoder without modifying the OpenGL application. This library intercepts application's glX$*$ calls and introduces NvIFROpenGL APIs in-between the application call and actual glX$*$ call in GL.

To use this library run an application using command line e.g. LD_PRELOAD=/path/to/this/library glxgears

OpenGL rendering into application Window will get redirected to FBO internally created and each frame is captured and encoded in glXSwapBuffers() call.

This library demonstrates the NvIFROpenGL capture and encode for single threaded applications using one OpenGL context and rendering into single Window.

2.31 GLIFRSimpleHwEnc

This sample shows how to use NvIFROGL to encode the contents of a FBO framebuffer with the hardware H.264 encoder. This is a simpler example showing how to use the SDK.

2.32 GLIFRSimplePBuffer

This sample shows how to initialize NvIFROGL and transfer the contents of a pbuffer to the host and write to a TGA bitmap file. A color 3D cube is drawn to the pbuffer.

2.33 GLIFRThreadedHwEnc

This sample shows how to use NvIFROGL to encode the contents of a FBO frame buffer with the hardware H.-264 encoder with multiple threads. One thread triggers the encoding while the other thread reads back the encoded bitstream.

The stream can be played back with the VLC media player (see www.videolan.org).

Notice

HDMI

OpenCL

Trademarks

Copyright