NVIDIA VIDEO CODEC SDK

Read Me

Table of Contents

Chapter 1. Read Me

1.1. Release Notes

What's new in Video Codec SDK 12.1?

Encode Features:

1. Explicit Split Frame Encoding: Enable/disable split frame encoding or use default automatic mode.
2. NVENC Low Level APIs for H.264, HEVC and AV1 encoders:
   - Iterative Encoding: Encode the same frame multiple times with different QP values and without advancing encoder state.
   - ReCon: Access to NVENC reconstructed frame.
   - NVENC Low Level Stats: Encoded frame output stats at row and block level.
   - External Lookahead: API to invoke lookahead explicitly.
3. CABR (Content Adaptive Bit Rate) acceleration using NVENC Low Level APIs.
4. Sample application to demonstrate accelerated file compression.

The C++ base classes in the SDK used for basic video encode and decode functionality are now released under MIT License instead of NVIDIA's End User License Agreement (EULA). Please check the license listed at the top of each individual source file.

Package Contents

This package contains the following:

1. Sample applications demonstrating various encoding/decoding/transcoding capabilities
   - [.\Samples\]
2. NVIDIA video encoder API header
   - [.\Interface\nvEncodeAPI.h]
3. NVIDIA video decoder API headers

- [.\Interface\cuviddec.h]
- [.\Interface\nvcuvid.h]

4. NVIDIA video decoder and encoder stub libraries
   - [.\Lib\linux\stubs\x86_64\libnvcuvid.so]
   - [.\Lib\linux\stubs\x86_64\libnvidia-encode.so]
   - [.\Lib\linux\stubs\ppc64le\libnvcuvid.so]
   - [.\Lib\linux\stubs\ppc64le\libnvidia-encode.so]
   - [.\Lib\linux\stubs\aarch64\libnvcuvid.so]
   - [.\Lib\linux\stubs\aarch64\libnvidia-encode.so]
   - [.\Lib\Win32\nvcuvid.lib]
   - [.\Lib\Win32\nvencodeapi.lib]
   - [.\Lib\x64\nvcuvid.lib]
   - [.\Lib\x64\nvencodeapi.lib]

The sample applications provided in the package are for demonstration purposes only and may not be fully tuned for quality and performance. Hence the users are advised to do their independent evaluation for quality and/or performance.

1.2. System Requirements

- NVIDIA Maxwell/Pascal/Volta/Turing/Ampere/Ada GPU with hardware video accelerators
  - Refer to the NVIDIA Video SDK developer zone web page (https://developer.nvidia.com/nvidia-video-codec-sdk) for GPUs which support video encoding and decoding acceleration.
- Video Codec SDK can be downloaded from https://developer.nvidia.com/nvidia-video-codec-sdk
- Video Codec SDK is available in GitLab at https://gitlab.com/nvidia/video/video-codec-sdk
- Documents can be browsed online at https://docs.nvidia.com/video-technologies/video-codec-sdk/index.html
- Windows: Driver version 531.61 or higher
- Linux: Driver version 530.41.03 or higher
- CUDA 11.0 or higher Toolkit
- Visual Studio Solution and Linux Makefiles can now be generated using CMake. CMake 3.9 or later is required for SDK 10.0 and higher. Self-extracting scripts or installers for CMake can be downloaded from https://cmake.org/download/.

Note: NVIDIA Video Codec SDK is now supported on IBM Power9 class server with NVIDIA Tesla V100 (SXM2) GPU.

Windows Configuration Requirements

- DirectX SDK is needed. You can download the latest SDK from Microsoft's DirectX website.
- The Vulkan SDK needs to be installed in order to build and run the AppMotionEstimationVkCuda sample application.
- To build sample applications that require FFmpeg headers and libraries, user must :
  - Get FFMPEG LGPL shared build (version 4.4) from BtbN repository. The package name is ffmpeg-n4.4-latest-win64-lgpl-shared-4.4.zip which can be found under Latest Auto-Build section in the BtbN github repository.
  - During cmake configuration phase (Section 1.3), set the FFMPEG_DIR cmake variable to point to the extracted FFMPEG directory containing the headers and libraries, i.e. the directory that contains bin, lib and include subdirectories.
    - `cmake -G"Visual Studio 16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DFFMPEG_DIR=<Path to FFMPEG directory> -DCMAKE_INSTALL_PREFIX=. ..`
  - If the user does not want to build any of the apps that depends on FFMPEG, then they can set cmake variable SKIP_FFMPEG_DEPENDENCY to TRUE during cmake configuration phase to skip setting up of FFMPEG libraries. This will exclude all the applications that depends on FFMPEG from the generated Visual Studio solution.
    - `cmake -G"Visual Studio 16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DSKIP_FFMPEG_DEPENDENCY=TRUE -DCMAKE_INSTALL_PREFIX=. ..`
  - If SKIP_FFMPEG_DEPENDENCY cmake variable is not set, then FFMPEG_DIR cmake variable must be set to point to the directory containing FFMPEG libraries and headers, else an error will be thrown during the cmake configuration phase.
- To build VideoCodecSDK applications that are dependent on freeglut and GLEW libraries, user must set the GLEW_DIR, GLUT_DIR and GLUT_INC cmake variables to point to appropriate directories containing binaries and headers for GLEW and freeglut libraries.
  - To setup GLEW libraries and headers required for the build, user must :
    - Get prebuilt GLEW binaries (version 2.1.0) from GLEW repository
    - During cmake configuration phase (Section 1.3), set the GLEW_DIR cmake variable to point to the extracted GLEW directory containing the headers and libraries, i.e. the directory that contains bin, lib and include subdirectories.
      - `cmake -G"Visual Studio 16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DGLEW_DIR=<Path to GLEW directory> -DCMAKE_INSTALL_PREFIX=. ..`
    - If the user does not want to build any application that depends on GLEW library, then they can set cmake variable SKIP_GL_DEPENDENCY to TRUE during cmake configuration phase to skip setting up of GLEW libraries required for the build. This will exclude applications that depends on GLEW from the generated Visual Studio solution.

- `cmake -G"Visual Studio 16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DSKIP_GL_DEPENDENCY=TRUE -DCMAKE_INSTALL_PREFIX=. ..`

- If the SKIP_GL_DEPENDENCY cmake variable is not set, then GLEW_DIR cmake variable must be set to point to the directory containing GLEW libraries and headers, else an error will be thrown during the cmake configuration phase.

- To setup freeglut libraries and headers required for the build, user must :
  - Get freeglut source code (version 3.4.0) from GLUT repository. The download link for version 3.4.0 can be found under Stable releases section.
  - Since freeglut does not distribute prebuilt libraries, user need to build the libraries from the fetched source code. Detailed instructions to build libraries from the freeglut source can be found in the README.cmake file in the freeglut source directory. Make sure to build the Release and Debug versions of freeglut as both are required for building VideoCodecSDK apps. After a successful build the lib/Debug subdirectory in freeglut build directory will have freeglut_staticd.lib and freeglutd.lib files, and the lib/Release subdirectory in freeglut build directory will have freeglut.lib and freeglut_static.lib files.
  - During cmake configuration phase (Section 1.3), set the GLUT_DIR cmake variable to point to the freeglut build directory containing the locally built libraries, i.e. the directory that contains bin and lib subdirectories.
  - Set the GLUT_INC cmake variable to point to the include directory in freeglut source directory, i.e. the directory that contain GL subdirectory which has the freeglut headers.
    - `cmake -G"Visual Studio16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DGLUT_DIR=<Path to freeglut build directory> -DGLUT_INC=<Path to freeglut include directory> -DCMAKE_INSTALL_PREFIX=. ..`
  - If the user does not want to build any application that depends on freeglut library, then they can set cmake variable SKIP_GL_DEPENDENCY to TRUE during cmake configuration phase to skip setting up of freeglut libraries required for the build. This will exclude applications that depends on freeglut from the generated Visual Studio solution.
    - `cmake -G"Visual Studio 16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DSKIP_GL_DEPENDENCY=TRUE -DCMAKE_INSTALL_PREFIX=. ..`
  - If the SKIP_GL_DEPENDENCY cmake variable is not set, then GLUT_DIR and GLUT_INC cmake variables must be set, else an error will be thrown during the cmake configuration phase.

- In order to build and run AppEncD3D12 sample application, Windows 20H1 or later is required. Visual Studio 2017 and above should be used for building and running this application. This application also requires Agility SDK. To configure Agility SDK :

- Download Agility SDK (D3D12SDKVersion 606 or later) and set the following cmake variables while generating the projects
  - AGILITY_SDK_BIN : to point to the directory containing the D3D12Core.dll for the platform.
  - AGILITY_SDK_VER : D3D12SDKVersion of the Agility SDK version used.

  AppEncD3D12 project will not be generated if the above cmake variables are not set.
- On building AppEncD3D12, D3D12 directory which contains the required dlls from Agility SDK is created along with AppEncD3D12.exe. Make sure to copy the D3D12 directory along with AppEncD3D12.exe if the executable is moved to some other location

- In Windows, the following environment variables must be set to build the sample applications included with the SDK
  - DXSDK_DIR: pointing to the DirectX SDK root directory.
  - VULKAN_SDK: pointing to Vulkan SDK install directory.
  - The CUDA Toolkit and the related environment variables are optional to install if the client has Video Codec SDK 8.0. However, they are mandatory if client has Video Codec SDK 8.1 or above on his/her machine.
- Plus all the requirements under System Requirements and Common to all OS platforms

Linux Configuration Requirements

- X11 and OpenGL, GLUT, GLEW libraries for video playback and display
- CUDA Toolkit is mandatory if client has Video Codec SDK 8.1 or above on his/her machine.
- Libraries and headers from the FFmpeg project which can be downloaded and installed using the distribution's package manager or compiled from source.
  - The sample applications have been compiled and tested against the libraries and headers from FFmpeg- 4.4. The source code of FFmpeg- 4.4 has been included in this SDK package. While configuring FFmpeg on Linux, it is recommended not to use 'disable-decoders' option. This configuration is known to have a channel error (XID 31) while executing sample applications with certain clips and/or result in an unexpected behavior.
- To build/use sample applications that depend on FFmpeg, users may need to
  - Add the directory (/usr/local/lib/pkgconfig by default) to the PKG_CONFIG_PATH environment variable. This is required by the Makefile to determine the include paths for the FFmpeg headers.
  - Add the directory where the FFmpeg libraries are installed to the LD_LIBRARY_PATH environment variable. This is required for resolving runtime dependencies on FFmpeg libraries.
- Stub libraries (libnvcuvid.so and libnvidia-encode.so) have been included as part of the SDK package, in order to aid development of applications on systems where the NVIDIA driver has not been installed. The sample applications in the SDK will link against these stub libraries

as part of the build process. However, users need to ensure that the stub libraries are not referenced when running the sample applications. A driver compatible with this SDK needs to be installed in order for the sample applications to work correctly.

- The Vulkan SDK needs to be installed in order to build and run the AppMotionEstimationVkCuda sample application.
- Plus all the requirements under System Requirements and Common to all OS platforms

Windows Subsystem for Linux (WSL) Configuration Requirements

- CUDA Toolkit is mandatory to use Video Codec SDK 8.1 and higher".
- Add the directory /usr/lib/wsl/lib to PATH environment variable, if not added by default. This is required to include path for the WSL libraries.
- Libraries and headers from the FFmpeg project which can be downloaded and installed using the distribution's package manager or compiled from source.
  - The sample applications have been compiled and tested against the libraries and headers from FFmpeg- 4.4. The source code of FFmpeg- 4.4 has been included in this SDK package. While configuring FFmpeg on WSL, it is recommended not to use 'disable-decoders' option. This configuration is known to have a channel error (XID 31) while executing sample applications with certain clips and/or result in an unexpected behavior.
- To build/use sample applications that depend on FFmpeg, users may need to
  - Add the directory (/usr/local/lib/pkgconfig by default) to the PKG_CONFIG_PATH environment variable. This is required by the Makefile to determine the include paths for the FFmpeg headers.
  - Add the directory where the FFmpeg libraries are installed to the LD_LIBRARY_PATH environment variable. This is required for resolving runtime dependencies on FFmpeg libraries.
- Plus all the requirements under System Requirements and Common to all OS platforms

Common to all OS platforms

- CUDA toolkit can be downloaded from http://developer.nvidia.com/cuda/cuda-toolkit
- Vulkan SDK can be downloaded from https://vulkan.lunarg.com/sdk/home. Alternatively, it can be installed by using the distribution's package manager.

1.3. Building Samples

Video Codec SDK uses CMake for building the samples. To build the samples, follow these steps:

Windows:

1. Install all dependencies for Windows, as specified in Windows Configuration Requirements

2. Extract the contents of the SDK into a folder.
3. Create a subfolder named "build" in Video_Codec_SDK_x.y.z/Samples
4. Open a command prompt in the "build" folder and run the following command, depending upon the version of Visual Studio on your computer.

   - Visual Studio 2019: `cmake -G"Visual Studio 16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=. ..`
   - Visual Studio 2017: `cmake -G"Visual Studio 15 2017" -A"x64" -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=. ..`
   - Visual Studio 2015: `cmake -G"Visual Studio 14 2015" -A"x64" -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=. ..`

   To build VideoCodecSDK applications having FFMPEG, freeglut or GLEW dependencies, make sure that required headers and libraries are setup as mentioned in Windows Configuration Requirements.

   For AppEncD3D12 project to be generated, cmake variables AGILITY_SDK_BIN and AGILITY_SDK_VER has to be set as mentioned in Windows Configuration Requirements. Add the following options to the above commands to set this cmake variables.

   - `-DAGILITY_SDK_BIN=<Path to Agility SDK folder containing D3D12Core.dll> -DAGILITY_SDK_VER=<D3D12SDKVersion of Agility SDK>`

   AppEncD3D12 project will not be generated if the above options are omitted.

This command will generate the necessary Visual Studio project files in the "build" folder. You can open NvCodec.sln file in Visual Studio and build. Alternatively, following command can be used to build the solution:

```
cmake --build . --target install --config Release
```

The application binaries will be available in Samples/build. Please note that the applications are validated only for x64 platform.

Linux:

1. Install all dependencies for Linux, as specified in Linux Configuration Requirements.
2. Extract the contents of the SDK into a folder.
3. Create a subfolder named "build" in Video_Codec_SDK_x.y.z/Samples
4. Use the following command to build samples in release mode.

   - `cmake -DCMAKE_BUILD_TYPE=Release ..`
   - `make`
   - `make install`

This will build and install the binaries of the sample applications. The application binaries will be available in the folder `Samples/build`.

Windows Subsystem for Linux:

1. Install all dependencies for Windows Subsystem for Linux, as specified in Windows Subsystem for Linux (WSL) Configuration Requirements.
2. Follow the build and installation steps provided above for Linux. Applications using OpenGL and Vulkan will not be built.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgment, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, CUDA Toolkit, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLProf, GPU, Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NVCaffe, NVIDIA Deep Learning SDK, NVIDIA Developer Program, NVIDIA GPU Cloud, NVLink, NVSHMEM, PerfWorks, Pascal, SDK Manager, Tegra, TensorRT, TensorRT Inference Server, Tesla, TF-TRT, Triton Inference Server, Turing, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com