NVIDIA OPTICAL FLOW SDK

Read Me

Table of Contents

Chapter 1. Read Me

1.1. System Requirements

▶ NVIDIA Turing and above GPUs - Refer to the NVIDIA Optical flow developer zone web page (https://developer.nvidia.com/opticalflow-sdk) for GPUs which support Optical flow and stereo disparity hardware acceleration.

▶ NVIDIA Optical flow SDK. It can be downloaded from https://developer.nvidia.com/opticalflow-sdk

▶ Windows: Driver version 528.24 or higher

▶ Linux: Driver version 525.85.05 or higher

Common to all OS platforms

▶ CUDA 10.2 or higher toolkit is required. It can be downloaded from http://developer.nvidia.com/cuda/cuda-toolkit

▶ CMake 3.14 or later. Self-extracting scripts or installers for CMake can be downloaded from https://cmake.org/download/.

▶ Vulkan SDK version 1.3.231 or later with the minimum required apiVersion is Vulkan 1.3. It can be downloaded from https://www.lunarg.com/vulkan-sdk/.

Windows Configuration Requirements

▶ Visual Studio 2017 and above along with latest Windows SDK

▶ Plus all the requirements under System Requirements and Common to all OS platforms

▶ Windows 10 or higher is required for Cuda, DirectX 11 and Vulkan interfaces

▶ Windows 10 20H1 or higher is required for DirectX 12 interface

Linux Configuration Requirements

▶ GCC 5.1 or newer is required to build and execute the sample applications.

▶ Building the sample applications from this SDK requires the FreeImage library to be installed. This version of the SDK has been tested against FreeImage 3.18.0. The

FreeImage interface is used to read image pairs for which the optical flow needs to be calculated. It is also used to generate flow-map of the flow vectors in *.png format.

End users can

▶ Install the library provided by their distribution. This is the recommended approach if the version of the distribution-provided library is the same as the one used for testing this SDK, or close to to it.

▶ Build and install the library from source. The source code for this library can be downloaded from http://freeimage.sourceforge.net/download.html. When compiling FreeImage for the PowerPC architecture, users must add the line `CFLAGS += -DPNG_POWERPC_VSX_OPT=0` to the Makefile.gnu file shipped as part of FreeImage, at the end of the existing set of lines which modify CFLAGS.

▶ Plus all the requirements under System Requirements and Common to all OS platforms

Windows Subsystem for Linux (WSL) Configuration Requirements

▶ Windows Subsystem for Linux is not supported with Vulkan interace.

▶ Configuration requirements for WSL are same as those mentioned for Linux under Linux Configuration Requirements.

▶ Additionally, directory /usr/lib/wsl/lib must be added to PATH environment variable, if not added by default. This is required to include path for the WSL libraries.

1.2.   Building Basic Samples

Optical Flow 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 Optical_Flow_SDK_x.y.z/NvOFBasicSamples
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=. ..`

This command will generate the necessary Visual Studio project files in the "build" folder. You can open NvOFSamples.sln file in Visual Studio and build.

Linux:

1. Install all dependencies for Linux, as specified in <u>Linux Configuration Requirements</u>.
2. Extract the contents of the SDK into a folder.
3. Create a directory named "build" in Optical_Flow_SDK_x.y.z/NvOFBasicSamples
4. Inside the "build" directory, use the following commands to build samples in release mode.

   ▶ `cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=. ..`

   ▶ `make`

   ▶ `sudo make install`

This will build and install the binaries of the sample applications. The binaries will be copied to the "build/bin/x64" directory.

<u>Windows Subsystem for Linux:</u>

1. Install all dependencies for WSL, as specified in <u>Windows Subsystem for Linux (WSL) Configuration Requirements</u>.
2. Follow the build and installation steps as specified in <u>Building Samples on Linux</u>.

1.3. NvOFTracker: Build and Run Instructions

1.3.1. Prerequisites

1. <u>CMake.</u> Version >= 3.14
2. Visual Studio for Windows 10. Visual Studio 2019 is recommended.
3. <u>CUDA.</u> Version = 11.8. For linux, the recommended installation mechanism is debian installation
4. <u>cuDNN.</u>Version = 8.6. For linux, the recommended installation mechanism is debian installation
5. <u>TensorRT.</u>Version = 8.5. For linux, the recommended installation mechanism is debian installation. Trtexec is generally found at /usr/source/tensorrt/bin for linux
6. <u>Video Codec SDK.</u> Version = 10.0
7. Git.
8. OpenCV. Refer OpenCV sub section in the Build sections

1.3.2. Windows 10 Build

Assume NvOFTracker is present here: `C:/Users/TestPC/Downloads/OpticalFlowSDK/NvOFTracker`. All paths below are relative to this path (unless specified otherwise)

CUDA, cuDNN, TensorRT(TRT)

Use the individual installation instruction for each of these libraries.

1. For cuDNN, copy each of the bin, lib and include folder contents to the corresponding folders in the cuda tool kit. This will let applications automatically search for cudnn header, libs and binaries as cuda toolkit is already in path

2. For TRT you can could do the same as above. If you choose not to, then add the lib folder (contains dlls) to path so that applications can find them at runtime.

Video Codec SDK

On downloading Video Codec SDK, if the folder VideoCodecSDK represents the root, then add `VideoCodecSDK/Samples/External/FFmpeg/lib/x64` to path so the necessary ffmpeg dlls are found by the application at run time.

OpenCV

Use the install script(`Scripts/installOCV_Windows.sh`) to install opencv. Note that you will need Git installed and you will need to run the installation script in Git bash. When all is done, there should be an install folder in the current directory. Go to `Scripts/Install/opencv/x64/vc14/bin` and copy the entire path and add it to your system Path variable. This will help applications find the opencv related dlls at run time.

NvOFTSample and NvOFTracker:

Steps to build:

1. Do, `cd C:/Users/TestPC/Downloads/OpticalFlowSDK/NvOFTracker && mkdir build && cd build`

2. Run, `cmake -DOpenCV_DIR=opencvDir -DTRT_ROOT=trtRoot -DVIDEOCODEC_SDK_ROOT=videocodecsdkroot ..`

   ▶ replace **opencvDir** with the directory containing OpenCVConfig.cmake (generally under `Scripts/Install/opencv` folder)

   ▶ replace **trtRoot** with the location of TensorRT root in your downloads (For eg. `C:/Users/TestPC/Downloads/TensorRT-8.x.y.z.Windows10.x86_64.cuda-11.p.cudnn8.q/TensorRT-8.x.y.z`)

   ▶ replace **videocodecsdk** with the location of Video Codec SDK root (necessarily the folder containing samples folder, `VideoCodecSDK/Samples`)

3. In the current directory there will be VS solution file with name `NvOFTrackerMain.sln`. Open it and build the INSTALL project.

4. The above will create a folder called bin. This folder will contain `nvoftracker.dll` library and `NvOFTSample` executable.

1.3.3.   Linux Build

Assume NvOFTRacker is present here: `/home/Downloads/OpticalFlowSDK/NvOFTracker`.
All paths below are relative to this path (unless specified otherwise)

CUDA, cuDNN, TensorRT

Use the individual installation instruction for each of these libraries. Use debian installation so
that all paths are configured.

Video Codec SDK

Unlike windows, the ffmpeg libraries need to be built for linux. You can find the source
of ffmpeg shipped as part of ffmpeg. If VideoCodecSDK is the root then `VideoCodecSDK/
Samples/External/FFmpeg/src` will contain the zipped src folder. Steps to build:

1. Unzip the source folder. cd into the folder.
2. `./configure --enable-shared`
3. `make -j 8`
4. `sudo make install`

This will install the ffmpeg libraries which then can be used by app.

OpenCV

Use the install script(`Scripts/installOCV_Linux.sh`) to install opencv. Make sure ffmpeg
is built before running this script. Please run `dos2unix installOCV_Linux.sh` in case there
are line ending related issues.

NvOFTSample and NvOFTracker:

Steps to build:

1. Do, `cd /home/Downloads/OpticalFlowSDK/NvOFTracker && mkdir build && cd
   build`
2. Run, `cmake -DOpenCV_DIR=opencvDir -
   DVIDEOCODEC_SDK_ROOT=videocodecsdkroot ..`

   ▶ replace **opencvDir** with the directory containing OpenCVConfig.cmake (generally under
      `Build/opencv` folder)

   ▶ replace **videocodecsdk** with the location of Video Codec SDK root (necessarily the
      folder containing samples folder, `VideoCodecSDK/Samples`)
3. In case you followed tar installation for TensorRT then Run, `cmake -
   DOpenCV_DIR=opencvDir -DVIDEOCODEC_SDK_ROOT=videocodecsdkroot -
   DTRT_ROOT=trtRoot ..`

- ▶ replace **opencvDir** with the directory containing OpenCVConfig.cmake (generally under `Build/opencv` folder)
- ▶ replace **videocodecsdk** with the location of Video Codec SDK root (necessarily the folder containing samples folder, `VideoCodecSDK/Samples`)
- ▶ replace **trtRoot** with the location of TensorRT root

4. Run `make install`
5. The above will create a folder called bin. This folder will container libnvoftracker.so library and NvOFTSample executable.

1.3.4.   Running applications

Building TensorRT Detector Engine:

You will need to build Tensorrt engine(.trt file) for the detector to be used in NvOFTSample. There is an onnx model of YOLOv3 detector at the below location:

`NvOFTSample/detector/models/yolov3.onnx`

you will need to use the above onnx to generate TRT engine. Note that onnx file is platform and GPU agnostic. But that is not the case of trt engine. TRT engine is specific to Operating System and GPU being used. **Steps**:

1. Navigate to directory containing trtexec.

   - ▶ For Windows, go to your TRT download location. Navigate to bin folder which contains trtexec.
   - ▶ For Linux, suggested method is to do `sudo find / -name trtexec`. This will spew the location. Generally it is under `/usr/src/tensorrt/bin`.

2. Use the following command to create the engine file.

   - ▶ Windows `trtexec --onnx=C:/Users/TestPC/Downloads/OpticalFlowSDK/ NvOFTracker/NvOFTSample/detector/models/yolov3.onnx -- saveEngine="yolov3.trt"`
   - ▶ Linux `trtexec --onnx=/home/Downloads/OpticalFlowSDK/NvOFTracker/ NvOFTSample/detector/models/yolov3.onnx --saveEngine="yolov3.trt"` Note that `--onnx yolov3.trt` will be created in the current directory. You can choose to provide some other location as well.

NvOFTSample

Run NvOFTSample to see the help menu. Use the engine generated above to run the samples. NvOFTSample only supports avi format for the –o parameter.

```
Mandatory Parameters
    -i          Input video file
    -e          TensorRT Engine file for Detector
    Optional Parameters:
    -o          Output video file
```

```
-fT          Filename to dump tracked objects
-dC          Dump tracked objects to Console
-sI          Detection skip interval. Must be 0 or greater
-g           GPU Id on which the tracker needs to run. Default is 0`
```

1.4.   NvFRUC: Build and Run Instructions

1.4.1.   Prerequisites

NVIDIA Optical Flow SDK v4.0 includes Frame Rate Up Conversion (FRUC) library that exposes FRUC APIs and source code of `NvOFFRUCSample` application.

NvOFFRUCSample application demonstrates use of FRUC APIs for frame rate up-conversion of video using NVIDIA optical flow at very high performance.

This section provides information on how to build and run NvOFFRUCSample application.

Common Prerequisites

▶ CMake 3.14 or later.

Self-extracting scripts or installers for CMake can be downloaded from https://cmake.org/download/.

Windows Prerequisites

▶ Visual Studio for Windows 10. Visual Studio 2017 is recommended.
▶ Visual Studio Redistributables
▶ Everything from Windows Configuration Requirements
▶ Windows OS should have latest updates.

Linux Prerequisites

▶ Supported distributions are Ubuntu18 and Ubuntu20.
▶ Everything from Linux Configuration Requirements

1.4.2.   Windows 10 Build

Assume `NvFRUCSample` is present at this path

`C:/Users/TestPC/Downloads/OpticalFlowSDK/NvFRUCSample`

All paths mentioned below are relative to the above path (unless specified otherwise).

Target directory for Windows build is following

`C:/Users/TestPC/Downloads/OpticalFlowSDK/NvFRUCSample/bin`

Above directory has following sub-directories that contain few required prebuilt dlls' as follows

▶ `win32`: Contains `NvFRUC.dll` and `cudart32_110.dll`
▶ `win64`: Contains `NvFRUC.dll` and `cudart64_110.dll`

Steps to build

1. To build the samples, first run the following command

   `cd C:/Users/TestPC/Downloads/OpticalFlowSDK/NvFRUCSample && mkdir build && cd build`

2. Run the following command

   ▶ For 64-bit build

     `cmake -G "Visual Studio 15 2017 Win64" ..`

   ▶ For 32-bit build

     `cmake -G "Visual Studio 15 2017" -A "Win32" ..`

   Please note ".." at the end of command. In case you haven't added cmake to environment variable Path, please follow instructions under "Windows" sub-section of Installing CMake.

   This will create Visual Studio solution files in the following directory

   `C:/Users/TestPC/Downloads/OpticalFlowSDK/NvFRUCSample/build`

3. Run the following command

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

   If 32-bit build was chosen, this step will copy 32-bit FreeImage.dll and NvFRUCSample.exe to `win32` directory else for 64-bit build it will copy 64-bit FreeImage.dll and NvFRUCSample.exe to `win64` directory.

   FreeImage.dll is necessary for the `NvFRUCSample` to run.

1.4.3.   Linux Build

Assume `NvFRUCSample` is present at this path

`/home/Downloads/OpticalFlowSDK/NvFRUCSample`

All paths mentioned below are relative to the above path (unless specified otherwise).

Target directory for Linux build is as follows

`/home/Downloads/OpticalFlowSDK/NvFRUCSample/bin`

Above directory has following sub-directories that contain few required prebuilt libraries as follows

▶ `ubuntu`: Contains `NvFRUC.so` and `libcudart.so.11.6.55`

Steps to build

1. Run the following command to create build directory

   ```
   cd /home/Downloads/OpticalFlowSDK/NvFRUCSample && mkdir build && cd build
   ```
2. Run the following command to create makefile

   ```
   cmake -DCMAKE_BUILD_TYPE=Release ..
   ```
3. Run the following command to create `NvFRUCSample` executable and install it in `ubuntu` directory

   ```
   make && make install
   ```

1.4.4.   Running applications

Open command prompt and `cd` to following directories

▶ **Windows**

   ▶ 32-bit

   ```
   C:/Users/TestPC/Downloads/OpticalFlowSDK/NvOFFRUC/NvOFFRUCSample/bin/
   win32
   ```

   ▶ 64-bit

   ```
   C:/Users/TestPC/Downloads/OpticalFlowSDK/NvOFFRUC/NvOFFRUCSample/bin/
   win64
   ```

▶ **Linux**

   ▶ `/home/Downloads/OpticalFlowSDK/NvOFFRUC/NvOFFRUCSample/bin/ubuntu`

Above directories have binary of `NvOFFRUCSample` application and all dependencies to run it.

If the application is being run on Linux OS, first you need to create symbolic link to libcudart as follows

1. cd to following directory where `NvOFFRUCSample` executable is built. In our example,

   ```
   cd /home/Downloads/OpticalFlowSDK/NvOFFRUC/NvOFFRUCSample/bin/ubuntu
   ```
2. `ln -s libcudart.so.11.6.55 libcudart.so.11.0`
3. `ln -s libcudart.so.11.0 libcudart.so`
4. `chmod +x NvOFFRUCSample`

If the application is being run in WSL you need to export path as follows

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/wsl/lib/:./
```

You can now run `NvOFFRUCSample` application from command prompt with appropriate command line.

Do not run the sample application executable with elevated permission.

To view help on supported command line parameters, run `NvOFFRUCSample` application without any command line parameters.

Here is snapshot showing help menu.

```
Mandatory Parameters
    --input             Input name , Path must be Absolute
    --width             Input frame width
    --height            Input frame height
    --output            Output Directory to store results,
                        Path could be Absolute or Relative
Optional Parameters
    --surfaceformat     Surface format : Set 0 for NV12, 1 for ABGR
    --startframe        Start processing from this frame index
    --endframe          Process till this frame index
    --allocationtype    Specify 0 to create CUDA and 1 to create DX allocations
    --cudasurfacetype   Specify 0 to create cuDevicePtr and
                        1 for cuArray, please note this option takes effect
                        only if client wants to use CUDA code path
```

Application accepts input video either as YUV file or as sequence of PNG frames. In case of PNG frames you need to follow some conventions in naming sequence of frames as described in example below.

Here are two examples of command line for Windows OS.

▶ **Input video as YUV file**

Path to input YUV file must not contain whitespaces.

In this case the application takes a video file in YUV420 format and interpolates intermediate frames between every two adjacent frames. It then interleaves these interpolated frames with original frames to generates output video. The output video thus generated has double the framerate as that of input. e.g. 30 fps to 60 fps.

Here is an example of command line:

```
NvOFFRUCSample.exe
--input=C:\fruc\inputfile.yuv
--width=1920
--height=1080
--output=C:\fruc\outputdir
```

Here inputfile.yuv is video file with frame width 1920 and frame height 1080."C:\fruc\outputdir" is the output directory where frame rate up converted YUV file will be saved. Application generates output file with name FRUC_(input file name).yuv. e.g. if input file name is inputfile.yuv, output would be named as "FRUC_inputfile.yuv".

▶ **Input video as sequence of PNG frames**

Path to input sequence of PNG frames is case sensitive.

In this case the application takes sequence of frames in PNG format, interpolates intermediate frames between every two frames and saves those in PNG format.

Here are few conventions you need to follow for naming sequence of PNG frames.

Names of all .png files should have a common string followed by a numeric value.

For example:

- rainbow_0001, rainbow_0002, rainbow_0003 and so on.
- flower01, flower02, flower03 and so on.

Number of digits in the numeric part should be the same for all the files.

If required you could pad 0's before the number.

You need to give "--input" parameter in following format <directory containing png files>\<common string in file name><*><.png>

In case of above examples you need to give path in the following way

- C:\wonders\rainbow_*.png
- C:\nature\flower*.png

Sample application interpolates the images and saves them in the folder mentioned in the "--output" parameter with the following names in case of the above example.

- rainbow_0001_interpolated, rainbow_0002_interpolated, rainbow_0003_interpolated and so on
- flower01_interpolated, flower02_interpolated, flower03_interpolated and so on

Here is an example of the command line that you could use in case of PNG frame sequence starting with name rainbow mentioned above.

```
NvOFFRUCSample.exe
--input=C:\fruc\wonders\rainbow_*.png
--width=1920
--height=1080
--output=C:\fruc\outputdir
```

In the example above, width of input frames is 1920 and height is 1080. Sample application will interpolate frames between every two frames and will save them in the folder "C:\fruc\outputdir" with naming convention as mentioned above.

1.4.5. Diagnostics

NvFRUCSample application generates log that can help with diagnostics. Log file logFRUCError.txt is created at following location.

- Windows

  "C:\ProgramData\NVIDIA Corporation\NvFRUC\logFRUCError.txt"

- Linux

  Log file ./NVFRUC/logFRUCError.txt will be created in the folder where NvFRUCSample binary got executed.

  Assume NvFRUCSample binary got executed from following location

  /home/Downloads/OpticalFlowSDK/NvFRUCSample/bin/ubuntu

  In this case log file would be created at following location

```
/home/Downloads/OpticalFlowSDK/NvFRUCSample/bin/ubuntu/NVFRUC/
logFRUCError.txt
```