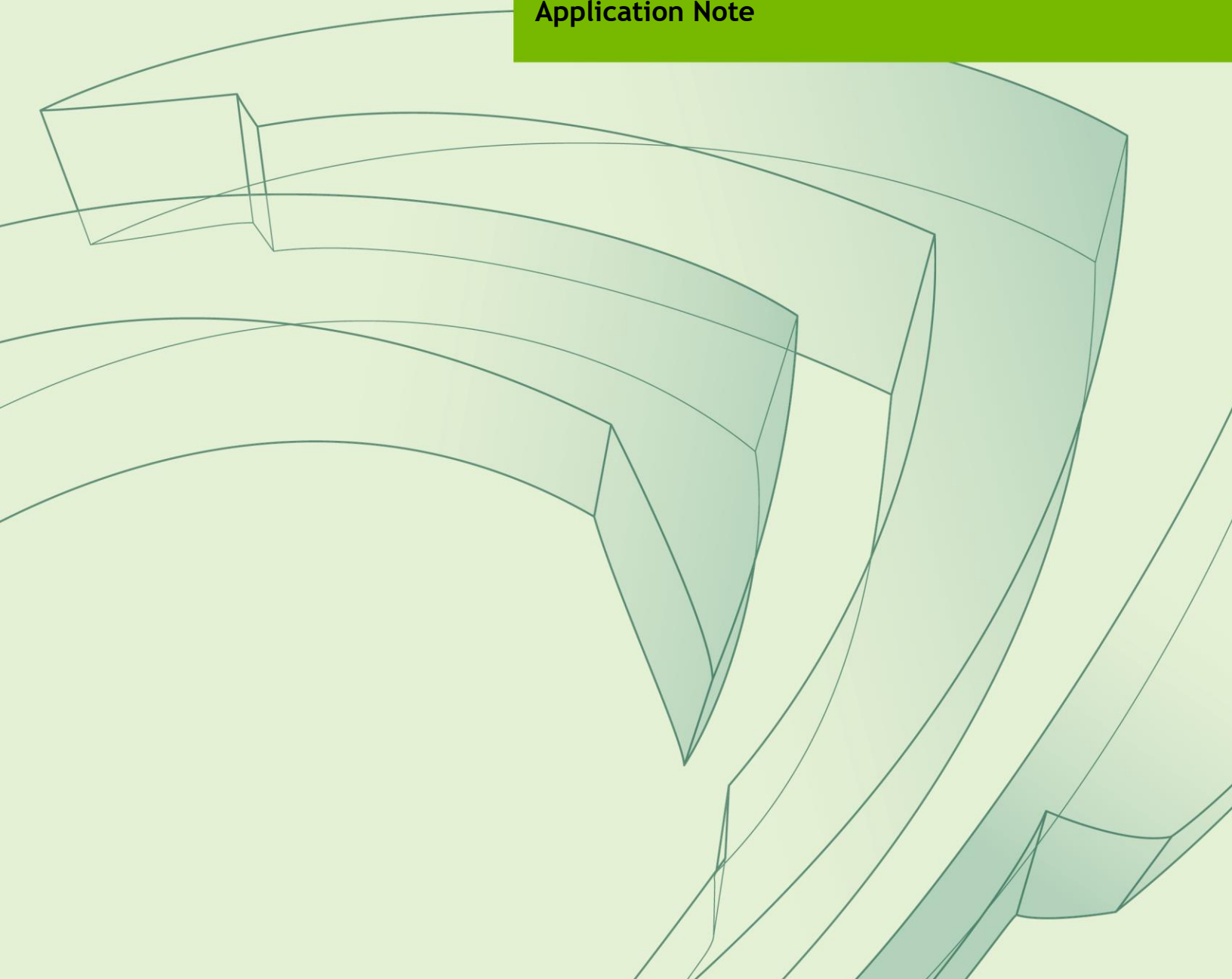
USING FFMPEG WITH NVIDIA GPU HARDWARE ACCELERATION

DA-08430-001_v01 | April 2017

Application Note



DOCUMENT CHANGE HISTORY

DA-08430-001_v01

Version	Date	Authors	Description of Change
1.0	2017/04/26	AP/YG	Initial release

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

All NVIDIA GPUs starting Kepler generation support fully accelerated hardware video encoding and all GPUs starting Fermi generation support fully accelerated hardware video decoding. The hardware encoder and hardware decoder are referred to as NVENC and NVDEC in the rest of the document.

The hardware capabilities of NVENC and NVDEC are exposed in NVIDIA's Video Codec SDK through APIs here-in referred to as NVENCODE API and NVDECODE API using which the user can access the hardware acceleration abilities of NVENC and NVDEC.

FFMPEG is the most popular multimedia transcoding software and is used extensively for video and audio transcoding.

NVENC and NVDEC can be effectively used with FFMPEG to significantly speed up video decoding, encoding, and end-to-end transcoding.

This document explains ways to accelerate video encoding, decoding and end-to-end transcoding on NVIDIA GPUs through FFMPEG which uses APIs exposed in NVIDIA Video Codec SDK.

Chapter 2. SETUP

2.1 HARDWARE SETUP

To run FFMPEG with NVIDIA GPU acceleration user needs a system with Linux or Windows operating system and a supported NVIDIA GPU.

For a list of supported GPUs, please refer to https://developer.nvidia.com/nvidia-video-codec-sdk.

For the rest of this document, it is assumed that the system being used has a GPU which has both NVENC and NVDEC.

2.2 SOFTWARE SETUP

2.2.1 Prerequisites

FFMPEG supports both Windows and Linux. FFMPEG has been compiled and tested with Microsoft Visual Studio 2013 SP2 and above (Windows), MinGW (msys2-x86_64-20161025) (Windows) and gcc 4.8 and above (Linux) compilers.

All the required files from NVIDIA Video SDK are included in the FFMPEG public repository. However, it is highly recommended to download the latest Video Codec SDK (although downloading the NVIDIA Video Codec SDK is not a prerequisite for compiling FFMPEG) and refer the documentation to understand the features and usage of APIs exposed for video encoding and decoding.

To compile FFMPEG, CUDA toolkit *must* be installed on the system, though CUDA toolkit is not needed to *run* the FFMPEG compiled binary.

Before using FFMPEG, it is recommended to refer to FFMPEG documentation and note down the version of Video Codec SDK it uses and ensure that the minimum driver required for that version of the Video Codec SDK is installed. For major releases of FFMPEG, Table 1 lists the minimum required driver versions.

Table 1. NVIDIA display driver requirement for running FFMPEG

FFMPEG Release	Video Codec SDK Version	Minimum required NVIDIA display driver version	
		Linux	Windows
v3.4	8.0	378.66	378.13
v3.3	7.0	367.44	369.05
v3.2	7.0	367.44	369.05

2.2.2 Compiling FFMPEG

FFMPEG is an open-source project. User needs to download the FFMPEG source code repository and compile it using an appropriate compiler.

More Information on building FFMPEG can be found at:
https://trac.ffmpeg.org/wiki/CompilationGuide

2.2.2.1 Compiling for Linux

FFMPEG with NVIDIA GPU acceleration is supported on all Linux platforms.

Following steps should be followed for compiling FFMPEG on Linux.

1. Clone FFMPEG's public GIT repository
```
git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg/
```

2. Install necessary packages
```
sudo apt-get install build-essential yasm cmake libtool libc6
libc6-dev unzip wget libnuma1 libnuma-dev
```

3. Configure
```
./configure --enable-nonfree --enable-nvenc -enable-cuda --
enable-cuvid -enable-libnpp --extra-cflags=-
I/usr/local/cuda/include --extra-ldflags=-L/usr/local/cuda/lib64
```

4. Compile
```
make -j 8
```

5. Install the libraries
```
sudo make install
```

2.2.2.2 Compiling for Windows

FFMPEG with NVIDIA GPU acceleration is supported on all Windows platforms, with compilation through Microsoft Visual Studio 2013 SP2 and above and MinGW.

Following steps should be followed for compiling FFMPEG on Windows.

1. Install msys2 from www.msys2.org.

2. Clone FFMPEG's public GIT repository.
```
git clone https://git.ffmpeg.org/ffmpeg.git
```

3. Create a folder named nv_sdk in the parent directory of FFMPEG and copy all the libraries from C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\include to and C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\lib\x64 to nv_sdk folder.

4. Launch the Visual Studio x64 Native Tools Command Prompt.

5. From the Visual Studio x64 Native Tools Command Prompt, launch the MinGW64 environment by running `mingw64.exe` from the msys2 installation folder.

6. In the MinGW64 environment, install the necessary packages.
```
pacman -S diffutil make pkg-config yasm
```

7. Add the following paths by running the commands.
```
export PATH="/c/Program Files (x86)/Microsoft Visual Studio
12.0/VC/BIN/amd64/":$PATH
export PATH="/c/Program Files/NVIDIA GPU Computing
Toolkit/CUDA/v8.0/bin/":$PATH
```

8. Go to the FFMPEG installation folder and run the following command.
```
./configure --enable-nonfree --enable-nvenc --enable-cuda --
enable-cuvid --enable-libnpp --extra-cflags=-Ilocal/include --
extra-cflags=-I../nv_sdk --extra-ldflags=-L../nv_sdk
```

9. Compile the code by executing the following command.
```
make -j 8
```

2.2.2.3 Readily Available FFMPEG Builds

Readily available builds for Windows having NVIDIA GPU acceleration enabled are available from the following site. This build does not have scale_npp filter by default, the scale_npp filter is included in the build when `--enable-nonfree` switch is turned on while configuring FFMPEG.

http://ffmpeg.zeranoe.com/builds/

2.2.2.4 Commonly faced Issues and tips to resolve them

- Common compilation issues
  1. FFMPEG TOT may be broken at times, please check out a release version if it is broken or use an older snapshot.
  2. Make sure you are using mingw64 for a 64-bit system, using mingw32 would result in errors like - "`Relocation truncated to fit – R_X86_64_32`".
  3. Msys (and not Msys2) does not have ability to launch mingw64 command shell but only mingw32 shell.
  4. Make sure cuda.h is available in /usr/include along with SDK header files, it is required for enabling NVCUVID, else configure will lead to an error - "`CUDA Not found`".
  5. Not specifying `--extra-ldflags` in correct format will lead to error - argument not recognized.
- Common run-time issues
  1. Use `-vsync 0` option with decode to prevent FFMPEG from creating output YUV with duplicate and extra frames.
  2. Msys2 gives errors such as - `"Libbz2-1.dll missing from your computer`" while running FFMPEG. Workaround for this error - Copy all DLLs under C:\msys64\mingw64\bin in the folder where ffmpeg.exe is present.

Chapter 3. BASIC TESTING

Once FFMPEG binary with NVIDIA hardware acceleration support is compiled, hardware accelerated video transcode should be tested to ensure everything works well.

3.1 1:1 HWACCEL TRANSCODE WITHOUT SCALING

The following command reads file *input.mp4* and transcodes it to *output.mp4* with H.264 video at the same resolution and with the same audio codec.

```
ffmpeg -y -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4 -c:a
copy -c:v h264_nvenc -b:v 5M output.mp4
```

3.2 1:1 HWACCEL TRANSCODE WITH SCALING

The following command reads file *input.mp4* and transcodes it to *output.mp4* with H.264 video at 720p resolution and with the same audio codec.

```
ffmpeg -y -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4 -vf
scale_npp=1280:720 -c:a copy -c:v h264_nvenc -b:v 5M output.mp4
```

3.3 1:5 HWACCEL TRANSCODE WITH SCALING

The following command reads file *input.mp4* and transcodes it to five different H.264 videos at various output resolutions and bit rates. Note that while using the GPU video encoder and decoder, this command also uses the scaling filter (*scale_npp*) in FFMPEG for scaling the decoded video output into multiple desired resolutions. Doing this ensures that the memory transfers (system memory to video memory and vice versa) are eliminated, and that transcoding is performed with the highest possible performance.

```
ffmpeg -y -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4
-vf scale_npp=1920:1080 -c:a copy -c:v h264_nvenc -b:v 8M
output_1080p.mp4
-vf scale_npp=1280:720 -c:a copy -c:v h264_nvenc -b:v 5M
output_720p.mp4
-vf scale_npp=640:480 -c:a copy -c:v h264_nvenc -b:v 3M h264_nvenc
output_480p.mp4
-vf scale_npp=320:240 -c:a copy -c:v h264_nvenc -b:v 2M h264_nvenc
output_240p.mp4
-vf scale_npp=160:128 -c:a copy -c:v h264_nvenc -b:v 1M h264_nvenc
output_128p.mp4
```

Chapter 4. QUALITY TESTING

Once basic FFMPEG setup is confirmed to be working properly, other options provided on the FFMPEG command line can be tried out to test encoding, decoding and transcoding.

The below sections list FFMPEG commands for accelerating video encoding, decoding and transcoding using NVENC and NVDEC.

4.1 VIDEO ENCODING

The quality of encoded video depends on various features in use by the encoder. To encode a 720p YUV, use the following command.

```
ffmpeg -y -vsync 0 –s 1280x720 –i input.yuv -c:v h264_nvenc output.mp4
```

This generates the output file in MP4 format (output.mp4) with H264 encoded video.

Video encoding can be broadly classified into two types of use cases:

- **Latency tolerant high quality:** In these kind of use cases latency is permitted. B-frames, variable bitrate (VBR) and higher VBV buffer sizes can be used. Typical use cases include cloud transcoding, recording and archiving etc.
- **Low latency:** In these kind of use cases latency should be low and can be as low as 16ms. In this mode, B-frames are disabled, constant bitrate modes are used and VBV-buffer sizes are kept very low. Typical use cases include real-time gaming, live streaming and video conferencing etc. This encoding mode however results to a lower encoding quality due to the above constraints.

NVENCODEAPI supports several features for adjusting quality, performance and latency which are exposed through FFMPEG command line. It is recommended to enable the feature(s)/command line option(s) depending on the use case.

4.2 VIDEO DECODING

The FFMPEG video decoder is straightforward to use. To decode an input bitstream from *input.mp4*, use the following command.

```
ffmpeg -y -vsync 0 -c:v h264_cuvid -i input.mp4 output.yuv
```

This generates the output file in YUV format (*output.yuv*).

4.3 COMMAND LINES FOR LATENCY-TOLERANT HIGH QUALITY TRANSCODING

4.3.1 1:1 HWACCEL Transcode without Scaling

Input: *input.mp4*

Output: *same resolution as input, bitrate = 5M (audio same as input)*

- Slow Preset

```
ffmpeg -y -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4 -c:a
copy  -c:v h264_nvenc -preset slow -profile high -b:v 5M -bufsize 5M -
maxrate 10M -qmin 0 -g 250 -bf 2 -temporal-aq 1 -rc-lookahead 20 -
i_qfactor 0.75 -b_qfactor 1.1 output.mp4
```

- Medium Preset

  Use `-preset medium` instead of `-preset slow` in above command line.

- Fast Preset

  Use `-preset fast` instead of `-preset slow` in above command line.

4.3.2 1:5 HWACCEL Transcode with Scaling

Input: *input.mp4*

Output: *1080p bitrate = 5M, 1080p bitrate=3M, 720p bitrate=5M, 720p bitrate=2M, 540p bitrate=1M (audio same as input)*

- Slow Preset

```
ffmpeg -y -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4
-vf scale_npp=1920:1080 -c:a copy -c:v h264_nvenc -preset slow -profile
high -b:v 5M -bufsize 5M -maxrate 10M -qmin 0 -g 250 -bf 2 -temporal-aq
1 -rc-lookahead 20 -i_qfactor 0.75 -b_qfactor 1.1 output_1080p_5M.mp4
-vf scale npp=1920:1080 -c:a copy -c:v h264 nvenc -preset slow -profile
high -b:v 3M -bufsize 3M -maxrate 6M -qmin 0 -g 250 -bf 2 -temporal-aq
1 -rc-lookahead 20 -i_qfactor 0.75 -b_qfactor 1.1 output_1080p_3M.mp4
-vf scale_npp=1280:720 -c:a copy -c:v h264_nvenc -preset slow -profile
high -b:v 5M -bufsize 5M -maxrate 10M -qmin 0 -g 250 -bf 2 -temporal-aq
1 -rc-lookahead 20 -i_qfactor 0.75 -b_qfactor 1.1 output_720p_5M.mp4
-vf scale_npp=1280:720 -c:a copy -c:v h264_nvenc -preset slow -profile
high -b:v 2M -bufsize 2M -maxrate 4M -qmin 0 -g 250 -bf 2 -temporal-aq
1 -rc-lookahead 20 -i_qfactor 0.75 -b_qfactor 1.1 output_720p_2M.mp4
-vf scale_npp=720:540 -c:a copy -c:v h264_nvenc -preset slow -profile
high -b:v 1M -bufsize 1M -maxrate 2M -qmin 0 -g 250 -bf 2 -temporal-aq
1 -rc-lookahead 20 -i_qfactor 0.75 -b_qfactor 1.1 output_540p_1M.mp4
```

- Medium Preset

  Use `-preset medium` instead of `-preset slow` in above command line

- Fast Preset

  Use `-preset fast` instead of `-preset slow` in above command line

4.4 COMMAND LINE FOR LOW LATENCY TRANSCODING

4.4.1 1:1 HWACCEL Transcode without Scaling

Input: *input.mp4 (30fps)*

Output: *same resolution as input, bitrate = 5M (audio same as input)*

- LLHQ (Low Latency High Quality) Preset

```
ffmpeg -y -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4 -c:a
copy  -c:v h264_nvenc -preset llhq -profile high -b:v 5M -bufsize 167K
-maxrate 5M -g 999999 -bf 0 output.mp4
```

- LLHP (Low Latency High performance) Preset

  Use `-preset llhp` instead of `-preset llhq` in above command line.

4.4.2 1:5 HWACCEL Transcode with Scaling

Input: *input.mp4 (30fps)*

Output: *1080p bitrate = 5M, 1080p bitrate=3M, 720p bitrate=5M, 720p bitrate=2M, 540p bitrate=1M (audio same as input)*

- LLHQ (Low Latency High Quality) Preset

```
ffmpeg -y -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4
-vf scale_npp=1920:1080 -c:a copy -c:v h264_nvenc -preset llhq -profile
high -b:v 5M -bufsize 167K -maxrate 5M -g 999999 -bf 0
output_1080p_5M.mp4
-vf scale_npp=1920:1080 -c:a copy -c:v h264_nvenc -preset llhq -profile
high -b:v 3M -bufsize 100K -maxrate 3M -g 999999 -bf 0
output_1080p_3M.mp4
-vf scale_npp=1280:720 -c:a copy -c:v h264_nvenc -preset llhq -profile
high -b:v 5M -bufsize 167K -maxrate 5M -g 999999 -bf 0
output_720p_5M.mp4
-vf scale_npp=1280:720 -c:a copy -c:v h264_nvenc -preset llhq -profile
high -b:v 2M -bufsize 67K -maxrate 2M -g 999999 -bf 0
output_720p_2M.mp4
-vf scale_npp=720:540 -c:a copy -c:v h264_nvenc -preset llhq -profile
high -b:v 1M -bufsize 33K -maxrate 1M -g 999999 -bf 0
output_540p_1M.mp4
```

- LLHP (Low Latency High Performance) Preset

  Use `-preset llhp` instead of `-preset llhq` in above command line.

Chapter 5. ADVANCED QUALITY SETTINGS

5.1 LOOKAHEAD

Lookahead improves the video encoder's rate control accuracy by enabling the encoder to buffer the specified number of frames, estimate their complexity, and allocate the bits appropriately among these frames proportional to their complexity. This typically results in better quality because the encoder can distribute the bits proportional to the complexity over a larger number of frames. The number of lookahead frames should be at least the number of B frames + 1 to avoid CPU stalls. A lookahead of 10-20 frames is suggested for optimal quality benefits.

To enable lookahead, use the `-rc-lookahead` *N* (*N* = number of frames) option on FFMPEG command line.

5.2 ADAPTIVE QUANTIZATION (AQ)

This feature improves visual quality by adjusting the encoding quantization parameter (QP) (on top of the QP evaluated by the rate control algorithm) based on spatial and temporal characteristics of the sequence. NVENC supports two flavors of AQ which are explained below. AQ internally uses CUDA for complexity estimation which may have slight impact on the performance and graphics engine utilization.

5.2.1 Spatial AQ

Spatial AQ mode adjusts QP values based on spatial characteristics of the frame. Since the low complexity flat regions are visually more perceptible to quality differences than high complexity detailed regions, extra bits are allocated to flat regions of the frame at

the cost of the regions having high spatial detail. Although Spatial AQ improves the perceptible visual quality of the encoded video, the required bit redistribution results in peak signal-to-noise ratio (PSNR) drop in most cases. Therefore, during PSNR-based evaluation, this feature should be turned off. The spatial AQ algorithm can be controlled by specifying the `aq-strength` parameter that controls the variations in QP values, with larger values bringing more QP variations. AQ strength ranges from 1-15.

To enable spatial AQ, please use `-spatial-aq 1` option on FFMPEG command line, and `-aq-strength 8` (can range from 1 to 15). If no value is specified the strength is auto selected by driver.

5.2.2 Temporal AQ

Temporal AQ tries to adjust encoding quantization parameter (QP) (on top of QP evaluated by the rate control algorithm) based on temporal characteristics of the sequence. Temporal AQ improves the quality of encoded frames by adjusting QP for regions which are constant or have low motion across frames but have high spatial detail, such that they become better reference for future frames. Allocating extra bits to such regions in reference frames is better than allocating them to the residuals in referred frames because it helps improve the overall encoded video quality. If majority of the region within a frame has little or no motion, but has high spatial details (e.g. high-detail non-moving background), enabling temporal AQ will benefit the most.

One of the potential disadvantages of temporal AQ is that enabling temporal AQ *may* result in high fluctuation of bits consumed per frame within a GOP. I/P-frames will consume more bits than average P-frame size and B-frames will consume lesser bits. Although target bitrate will be maintained at the GOP level, the frame size will fluctuate from one frame to next within a GOP more than it would without temporal AQ. If a strict CBR profile is required for every frame size within a GOP, it is not recommended to enable temporal AQ. To enable temporal AQ, please use `-temporal_aq 1` option on FFMPEG command line.
Temporal AQ is supported only for H.264.

Chapter 6. PERFORMANCE EVALUATION

To measure GPU aggregate performance, please follow the steps explained below:

- Run multiple simultaneous sessions (say 4 FFMPEG sessions) in parallel, each performing transcoding.
- Ensure the inputs have large number of frames (more than 15 seconds of video is recommended) so that initialization time overhead can be ignored.
- Measure the time required by each transcode.
- Derive the performance in terms of frames per second (FPS).

Notice

HDMI

OpenCL

Trademarks

Copyright