VCodecLibav C++ library

v1.1.0

Table of contents

- Overview
- Versions
- Library files
- VCodecLibav class description
  - VCodecLibav class declaration
  - getVersion method
  - transcode method
  - setParam method
  - getParam method
  - executeCommand method
- Build and connect to your project
- Simple example

Overview

VCodecLibav C++ library provides video encoding and decoding functions for H264, HEVC(H265) and JPEG codecs for Linux OS based on FFmpeg. The library supports software encoders / decoders and Intel hardware encoders / decoders. The library compatible with any CPU which supports FFmpeg. **VCodecLibav** video codec class inherits interface and data structures form VCodec interface library. The library depends on open source VCodec library (provides codec interface, Apache 2.0 license), open source Libav library (part of FFmpeg) and open source Logger (provides logging functions, Apache 2.0). Used FFmpeg codecs:

Codec	ffmpeg encoder	ffmpeg decoder
H264	**h264_vaapi** (Intel VAAPI based codec) for hardware encoding or **libx264** for software encoding.	h264_qsv (Intel QuickSync based) for hardware decoding or **libx264** for software decoding.

Codec	ffmpeg encoder	ffmpeg decoder
HEVC (H265)	**hevc_vaapi** (Intel VAAPI based codec) for hardware encoding or **libx265** for software encoding.	hevc_qsv (Intel QuickSync based) for hardware decoding or **libx265** for software decoding.
JPEG	**mjpeg_vaapi** (Intel VAAPI based codec) for hardware encoding and **mjpeg** for software encoding. **Warning: mjpeg** software codec produces not fully compatible JPEG (doesn't have DHT like **JPEG** does).	mjpeg_qsv (Intel QuickSync based) for hardware decoding and **mjpeg** for software decoding.

The primary difference between software (SW) and hardware (HW) encoders and decoders is the latency they introduce. SW encoders and decoders have **zero frame latency**, but they take more time than HW encoders and decoders. On the other hand, HW encoders have **1 frame latency**, and HW decoders have **3 frames latency**. Encoding/decoding time on 11th Gen Intel(R) Core(TM) i5-1145G7E @ 2.60GHz, Ubuntu 22.10, msec:

Hardware codec / resolution	1920x1080	1280x720	640x512
H264	5 msec / 3 msec	2 msec / 1.6 msec	0.6 msec / 0.5 msec
H265	5 msec / 3 msec	3 msec / 1.2 msec	2 msec / 0.6 msec
JPEG	2 msec / 4 msec	1 msec / 1.5 msec	0.3 msec / 0.6 msec
Software codec (1 thread) / resolution	**1920x1080**	**1280x720**	**640x512**
H264	8 msec / 3 msec	3.5 msec / 1.5 msec	1.8 msec / 0.9 msec
H265	35 msec / 10 msec	25 msec / 8 msec	15 msec / 7 msec
JPEG	4 msec / 3 msec	1.8 msec / 1.4 msec	0.9 msec / 0.7 msec

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	29.06.2023	First version.
1.0.1	02.07.2023	- Source code cleaned up. - Documentation updated.

Version	Release date	What's new
1.0.2	16.11.2023	- VCodec class updated.
1.0.3	20.12.2023	- Add specific encoder settings for noisy video. - Example added.
1.1.0	17.01.2024	- Add software encoder\decoder support for JPEG, H264 and HEVC. - Examples updated.

Library files

The library supplied by source code only. The user would be given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ---------------- Main CMake file.
README.md --------------------- Documentation.
3rdparty ---------------------- Folder with 3rdparty libraries.
    CMakeLists.txt ------------ CMake file for to include 3rdparty libraries.
    VCodec -------------------- Folder with VCodec interface library files.
    Logger -------------------- Folder with Logger library files.
src --------------------------- Folder with library source code.
    CMakeLists.txt ------------ CMake file.
    VCodecLibav.h ------------- Main library header file.
    VCodecLibavVersion.h ------ Header file with library version.
    VCodecLibavVersion.h.in --- File for CMake to generate version header.
    VCodecLibav.cpp ----------- C++ implementation file.
test -------------------------- Folder for test application files
    CMakeLists.txt ------------ CMake file for test application.
    main.cpp ------------------ Source C++ file of test application.
example ----------------------- Folder for simple example.
    CMakeLists.txt ------------ CMake file of example.
    main.cpp ------------------ Source C++ file of example.
```

Additionally, demo application depends on open source OpenCV library to provide video source read and file write functions.

VCodecLibav class description

VCodecLibav class declaration

VCodecLibav class declared in **VCodecLibav.h** file. Class declaration:

```
class VCodecLibav : public VCodec
{
public:
```

```
    /// Get library version.
    static std::string getVersion();

    /// Class constructor.
    VCodecLibav();

    /// Class destructor.
    ~VCodecLibav();

    /// Encode video frame.
    bool transcode(Frame& src, Frame& dst);

    /// Set parameter value.
    bool setParam(VCodecParam id, float value);

    /// Get parameter value.
    float getParam(VCodecParam id);

    /// Execute command.
    bool executeCommand(VCodecCommand id);
};
```

getVersion method

getVersion() method returns string of current version of **VCodecLibav** class. Method declaration:

```
static std::string getVersion();
```

Method can be used without **VCodecLibav** class instance:

```
std::cout << "VCodecLibav class version: " << VCodecLibav::getVersion() << std::endl;
```

Console output:

```
VCodecLibav class version: 1.1.0
```

transcode method

transcode(...) method intended to encode and decode video frame (Frame class). Video codec encodes/decodes video frames frame-by-frame. Method declaration:

```
bool transcode(Frame& src, Frame& dst);
```

Parameter	Value
src	Source video frame (see Frame class description). To encode video data **src** frame must have **NV12** pixel format. To decode video data **src** frame must have compressed pixel format (field **fourcc** of Frame class): **JPEG**, **H264** or **HEVC**.

Parameter	Value
dst	Result video frame (see Frame class description). To decode video data **src** frame must have compressed pixel format (field **fourcc** of Frame class): **JPEG**, **H264** or **HEVC**. User must specify output pixel format in advance. In case decoding video codec will set **NV12** pixel format automatically.

Returns: TRUE if frame was encoded/decoded or FALSE if not.

setParam method

setParam(...) method designed to set new video codec parameters value. Method declaration:

```
bool setParam(VCodecParam id, float value);
```

Parameter	Description
id	Video codec parameter ID according to **VCodecParam** enum.
value	Video codec parameter value. Valid values depends on parameter ID.

Returns: TRUE is the parameter is set or FALSE if not.

VCodec.h file of VCodec library defines IDs for parameters (**VCodecParam** enum) and IDs for commands (**VCodecCommand** enum). VCodecParam declaration:

```
enum class VCodecParam
{
    /// [read/write] Log level:
    /// 0-Disable, 1-Console, 2-File, 3-Console and file.
    LOG_LEVEL = 1,
    /// [read/write] Bitrate, kbps. For H264 and H265 codecs.
    BITRATE_KBPS,
    /// [read/write] Quality 0-100%. For JPEG codecs.
    QUALITY,
    /// [read/write] FPS. For H264 and H265 codecs.
    FPS,
    /// [read/write] GOP size. For H264 and H265 codecs.
    GOP,
    /// [read/write] H264 profile: 0 - Baseline, 1 - Main, 2 - High.
    H264_PROFILE,
    /// [read/write] Codec type. Depends on implementation.
    TYPE,
    /// Custom 1. Thread's count for software encoder\decoder. Depends on implementation.
    CUSTOM_1,
    /// Custom 2. Depends on implementation.
    CUSTOM_2,
    /// Custom 3. Depends on implementation.
    CUSTOM_3
};
```

Table 2 - Video codec params description. Some params not supported by particular VCodecLibav library.

Parameter	Access	Description
LOG_LEVEL	read / write	Logging mode. Values: **0** - Disable. **1** - Only file. **2** - Only terminal (console). **3** - File and terminal.
BITRATE_KBPS	read / write	Bitrate, kbps. For H264 and H265(HEVC) encoding. According to this value, FPS and GOP size video codec calculate parameter for H264 or H265(HEVC) encoding. **- Sets special settings for noisy video if bitrate is 0 for hardware encoding.** **- For software JPEG encoding bitrate incising gives higher quality**
QUALITY	read / write	Quality **0**(low quality)-**100%**(maximum quality). **Only** for **hardware JPEG** encoding. **Not supported** by JPEG SW encoding.
FPS	read / write	FPS. For H264 and H265(HEVC) encoding only. According to this value, FPS and GOP size video codec calculate parameter for H264 and H265(HEVC) encoding.
GOP	read / write	GOP size (Period of key frames) for H264 and H265(HEVC) encoding. Value: **1** - each output frame is key frame, **20** - each 20th frame is key frame etc.
H264_PROFILE	read / write	H264 profile for H264 encoding: **0** - Baseline. **1** - Main. **2** - High. For **hardware** encoding only.
TYPE	read / write	Type of encoder/decoder: **0** - hardware (default). **1** - software.
CUSTOM_1	read / write	Number of thread for software encoder/decoder. Thread's count for software encoder\decoder **H264 / HEVC(H265)** (not for JPEG): **1 - 32**, default value **1**. **Note:** in case one thread software encoder produces only one slice per frame. In case multiple threads frame can have multiple slices which can effect on RTSP servers.
CUSTOM_2	read / write	**Not supported** by VCodecLibav library.
CUSTOM_3	read / write	**Not supported** by VCodecLibav library.

getParam method

getParam(...) method designed to obtain video codec parameter value. Method declaration:

```
float getParam(VCodecParam id);
```

Parameter	Description
id	Video codec parameter ID according to **VCodecParam** enum (see Table 2).

Returns: parameter value or -1 of the parameters doesn't exist in particular video codec class.

executeCommand method

executeCommand(...) method designed to execute video codec command. Version **1.1.0** doesn't support commands. Method will return FALSE. Method declaration:

```
bool executeCommand(VCodecCommand id);
```

Parameter	Description
id	Video codec command ID according to **VCodecCommand** enum.

Returns: method returns FALSE in any case.

VCodec.h file of VCodec library defines IDs for parameters (**VCodecParam** enum) and IDs for commands (**VCodecCommand** enum). VCodecCommand declaration:

```
enum class VCodecCommand
{
    /// Reset.
    RESET = 1,
    /// Generate key frame. For H264 and H265 codecs.
    MAKE_KEY_FRAME
};
```

Table 3 - Video codec commands description. Some commands maybe unsupported by particular video codec class.

Command	Description
RESET	**Not supported** by VCodecLibav.
MAKE_KEY_FRAME	**Not supported** by VCodecLibav.

Build and connect to your project

Before compiling you have to install **ffmpeg** packages and additional applications for your system. For Ubuntu it's:

```
sudo apt-get install -y build-essential cmake ffmpeg libavcodec-dev libavutil-dev
libavformat-dev libavdevice-dev libavfilter-dev libcurl4
```

To compile demo application and examples install OpenCV library:

```
sudo apt-get install -y libopencv-dev
```

Typical commands to build **VCodecLibav** library:

```
cd VCodecLibav
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want connect **VCodecLibav** library to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

Create folder **3rdparty** in your repository and copy **VCodecLibav** repository folder to **3rdparty** folder. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    VCodecLibav
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

###############################################################################
## 3RD-PARTY
## dependencies for the project
###############################################################################
```

```
project(3rdparty LANGUAGES CXX)

################################################################################
## SETTINGS
## basic 3rd-party settings before use
################################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

################################################################################
## CONFIGURATION
## 3rd-party submodules configuration
################################################################################
SET(${PARENT}_SUBMODULE_VCODEC_LIBAV                    ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VCODEC_LIBAV)
    SET(${PARENT}_VCODEC_LIBAV                          ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VCODEC_LIBAV_TEST                     OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VCODEC_LIBAV_EXAMPLE                  OFF CACHE BOOL "" FORCE)
endif()

################################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
################################################################################
if (${PARENT}_SUBMODULE_VCODEC_LIBAV)
    add_subdirectory(VCodecLibav)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **VCodecLibav** to your project. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    VCodecLibav
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include VCodecLibav library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} VCodecLibav)
```

Done!

Simple example

Simple example opens video file with OpenCV, captures video from file, convert frame to NV12 pixel format (required for codec) and encodes. Source code:

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "VCodecLibav.h"

int main(void)
{
    // Create codec object and set params.
    cr::video::VCodec* encoder = new cr::video::VCodecLibav();
    encoder->setParam(cr::video::VCodecParam::BITRATE_KBPS, 10000);
    encoder->setParam(cr::video::VCodecParam::FPS, 20);
    encoder->setParam(cr::video::VCodecParam::GOP, 30);
    encoder->setParam(cr::video::VCodecParam::H264_PROFILE, 0);

    // Open video file with OpenCV.
    cv::VideoCapture videoSource;
    if (!videoSource.open("test.mp4"))
        return -1;

    // Get frame size from video source.
    int width = (int)videoSource.get(cv::CAP_PROP_FRAME_WIDTH);
    int height = (int)videoSource.get(cv::CAP_PROP_FRAME_HEIGHT);

    // Init frames.
    cv::Mat inputFrameBgr(height, width, CV_8UC3);
    cv::Mat inputFrameYuv(height, width, CV_8UC3);
    cr::video::Frame nv12Frame(width, height, cr::video::Fourcc::NV12);
    cr::video::Frame h264Frame(width, height, cr::video::Fourcc::H264);

    // Main loop.
    while (true)
    {
        // Capture next video frame.
        videoSource >> inputFrameBgr;
        if (inputFrameBgr.empty())
        {
            // Set first video frame position.
            videoSource.set(cv::CAP_PROP_POS_FRAMES, 1);
            continue;
        }

        // Convert BGR to YUV.
        cvtColor(inputFrameBgr, inputFrameYuv, cv::COLOR_BGR2YUV);

        // Convert YUV to NV12 (replacing pixels). You can use something else.
        size_t p = height;
        nv12Frame.frameId++; // Just to show unique info.
        for (size_t i = 0; i < (size_t)height; i = i + 2)
        {
```

```
            for (size_t j = 0; j < (size_t)width; j = j + 2)
            {
                nv12Frame.data[i * (size_t)width + j] =
                inputFrameYuv.data[i * (size_t)width * 3 + j * 3];
                nv12Frame.data[i * (size_t)width + j + 1] =
                inputFrameYuv.data[i * (size_t)width * 3 + j * 3 + 3];
                nv12Frame.data[(i + 1) * (size_t)width + j] =
                inputFrameYuv.data[(i + 1) * (size_t)width * 3 + j * 3];
                nv12Frame.data[(i + 1) * (size_t)width + j + 1] =
                inputFrameYuv.data[(i + 1) * (size_t)width * 3 + j * 3 + 3];
                nv12Frame.data[p * width + j] =
                inputFrameYuv.data[i * (size_t)width * 3 + j * 3 + 1];
                nv12Frame.data[p * width + j + 1] =
                inputFrameYuv.data[i * (size_t)width * 3 + j * 3 + 2];
            }
            ++p;
        }

        // Encode data.
        if (!encoder->transcode(nv12Frame, h264Frame))
        {
            std::cout << "Can't encode frame" << std::endl;
            continue;
        }

        // Show info.
        std::cout << "[" << h264Frame.frameId << "] Size " <<
        h264Frame.size << " Compression ratio : %" <<
        (int)(100.0f * ((float)h264Frame.size / (float)nv12Frame.size)) <<
        std::endl;
    }
}
```

Test application

Test application (VCodecLibav/test/main.cpp) for **VCodecLibav** C++ library shows how library works. The test application generates an artificial video, compresses it based on user-defined parameters such as codec type, bitrate, JPEG quality, GOP size, and H264 profile, and writes the results to a binary file named “out.h264”, “out.hevc”, or “out.jpeg”. The compressed frame is then decoded, and the generated frame and the frame after encoding and decoding are displayed if it set. To run application perform commands:

```
cd <application folder>
sudo chmod +x VCodecLibav
sudo ./VCodecLibav
```

After start you will see output:

```
===================================
VCodecLibav v1.1.4 test
===================================
```

```
Enter Encoder/Decoder type (0 - JPEG, 1 - H264, 2 - HEVC) : 0
Enter implementation type (0 - HW, 1 - SW) : 1
Default params:
Bitrate, kbps 60000
FPS: 30
GOP size: 30
Video width 1920
Video height 1080
Use default params (0 - no, 1 - yes) : 1
Show video (0 - no, 1 - yes) ? :
```

During encoding and decoding the application shows encoded data size and encoding/decoding time:

```
Data size 202718/3110400  encoding time msec  5.493 ||  4.276 msec decoding time
Data size 180286/3110400  encoding time msec  4.876 ||  3.608 msec decoding time
Data size 212329/3110400  encoding time msec  5.086 ||  4.177 msec decoding time
Data size 164472/3110400  encoding time msec  4.262 ||  3.773 msec decoding time
Data size 215343/3110400  encoding time msec  5.287 ||  3.916 msec decoding time
Data size 206060/3110400  encoding time msec  4.513 ||  5.234 msec decoding time
Data size 166682/3110400  encoding time msec  3.834 ||  3.228 msec decoding time
```

If you choose show video press **ESC** to exit test application.