JORDI CENZANO

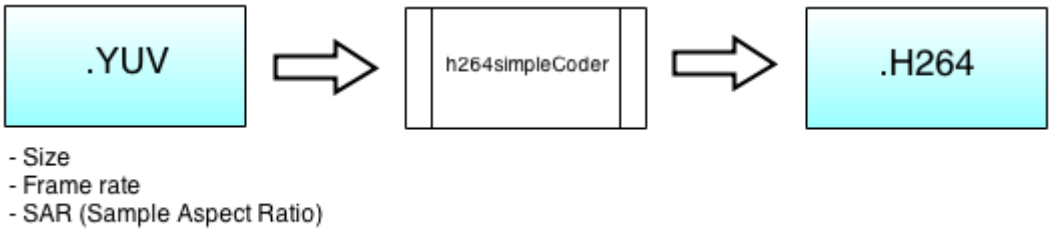
PERSONAL PAGE

A minimal h264 “encoder” (C++)

31/08/2014 6 COMMENTS (HTTPS://JORDICENZANO.NAME/2014/08/31/THE-SOURCE-CODE-OF-A-MINIMAL-H264-ENCODER-C/#COMMENTS)

Introduction

I always thought that the best approach to complex things is to start from the basis, and once you control this part enter step by step towards more complex parts. With this idea in mind and the aim to improve my understanding of h264 bitstream and its techniques I have created the following code in C++ that generates a compliant h264 file from a YUV420p file.



(https://jordicenzano.files.wordpress.com/2014/08/h264minencoderfig1-1.png)

Figure 1: Simple block diagram

First of all I have to say that the following source code is NOT a h264 encoder but it generates a compliant (playable) h264 stream, this means that the size of output file will be slightly bigger than the size of the input file. This it is because I have used only I_PCM macroblock codification (non compressed format) to describe the images in h264 stream.

I think that to read and understand this code could be a good starting point to start flirting with the h264 standard, better than dive into the standard (more than 700 pages of dense text).

References

- World’s smallest H.264 encoder By Ben Mesander (http://www.cardinalpeak.com/blog/worlds-smallest-h-264-encoder/)
  - Great article that implements a simple h.264 coder with hard coded headers (there is a bug in slice header)
- ITU-T H.264 standart (04/2013) (https://www.itu.int/rec/T-REC-H.264)

- The complete h264 standard

Input files

- Size:
  - **Multiple of 16 in height and width**
- Pixel format:
  - **yuv420p**
- Frame rate:
  - Any
- Aspect ratio:
  - Any

To create a compliant input file you can use ffmpeg (https://www.ffmpeg.org/), here you have 2 examples. The fisrt example convert any video to a YUV420p 128×96 file.

```
ffmpeg.exe -i anyvideo.avi -s 128x96 -pix_fmt yuv420p out.yuv
```

The second example generates a yuv420p blank (green bck) video file of 10secs, 128×96 pixels ,and 25fps (10s x 25fps = 250 progressive YUV frames):

```
ffmpeg.exe -t 10 -s 128x96 -f rawvideo -pix_fmt yuv420p -r 25 -i /dev/zero
```

Using the h264 simple coder

To use this h264 basic coder is very easy, just follow these steps:

1. Open a YUV420p format file
2. Open the destination file
3. Create an instance of CJOCh264encoder (passing the parameter of destination file pointer)
4. Call IniCoder with the following parameters:
   - nImW: Frame width in pixels
   - nImH: Frame height in pixels
   - nFps: Desired frames per second of the output file (typical values are: 25, 30, 50, etc)
   - SampleFormat: Sample format of the input file. In this implementation only SAMPLE_FORMAT_YUV420p is allowed
   - nSARw Indicates the horizontal size of the sample aspect ratio (typical values are:1, 4, 16, etc)
   - nSARh Indicates the vertical size of the sample aspect ratio (typical values are:1, 3, 9, etc)
5. Over all frames in the input file:
   - Get the frame data pointer calling GetFramePtr
   - Load a new frame from source file over the pointer returned by GetFramePtr
   - Call CodeAndSaveFrame to code the frame and save it to destination file
6. Finally, call CloseCoder and close the opened files

If you compile the h264simpleCoder.cpp, you could call the resulting application using this expression:

```
h264simpleCoder AVsyncTest.yuv OutTest.h264 128 96 25 16 9
```

The following source code shows how to use the CJOCh264encoder class including error handling.

```
//===================================================================
// Name        : h264simpleCoder.cpp
// Author      : Jordi Cenzano (www.jordicenzano.name)
// Version     : 1.0
// Copyright   : Copyright Jordi Cenzano 2014
// Description : Simple h264 encoder
//===================================================================

#include <iostream>
#include "CJOCh264encoder.h"

using namespace std;

int main(int argc, char **argv)
{
        int nRc = 1;

        puts("Simple h264 coder by Jordi Cenzano (www.jordicenzano.nam
        puts("This is NOT a video compressor, only uses I_PCM macroblo
        puts("It is made only for learning purposes");
        puts("*****************************************************

        if (argc < 3)
        {
                puts("-----------------------------------------------
                puts("Usage: h264mincoder input.yuv output.h264 [image
                puts("Default parameters: Image width=128 Image height
                puts("Assumptions: Input file is yuv420p");
                puts("-----------------------------------------------
                return nRc;
        }

        char szInputFile[512];
        char szOutputFile[512];
        int nImWidth = 128;
        int nImHeight = 96;
        int nFps = 25;
        int nSARw = 1;
        int nSARh = 1;

        //Get input file
        strncpy (szInputFile,argv[1],512);

        //Get output file
        strncpy (szOutputFile,argv[2],512);

        //Get image width
        if (argc > 3)
        {
                nImWidth = (int) atol (argv[3]);
```

```
        if (nImWidth == 0)
                puts ("Error reading image width input paramet
}

//Get image height
if (argc > 4)
{
        nImHeight = (int) atol (argv[4]);
        if (nImHeight == 0)
                puts ("Error reading image height input parame
}

//Get fps
if (argc > 5)
{
        nFps = (int) atol (argv[5]);
        if (nFps == 0)
                puts ("Error reading fps input parameter");
}

//Get SARw
if (argc > 6)
{
        nSARw = (int) atol (argv[6]);
        if (nSARw == 0)
                puts ("Error reading AR SARw input parameter")
}

//Get SARh
if (argc > 7)
{
        nSARh = (int) atol (argv[7]);
        if (nSARh == 0)
                puts ("Error reading AR SARh input parameter")
}

FILE *pfsrc = NULL;
FILE *pfdst = NULL;

pfsrc = fopen (szInputFile,"rb");
if (pfsrc == NULL)
{
        puts ("Error opening source file");
        return nRc;
}

pfdst = fopen (szOutputFile,"wb");
if (pfdst == NULL)
{
        puts ("Error opening destination file");
        return nRc;
}
```

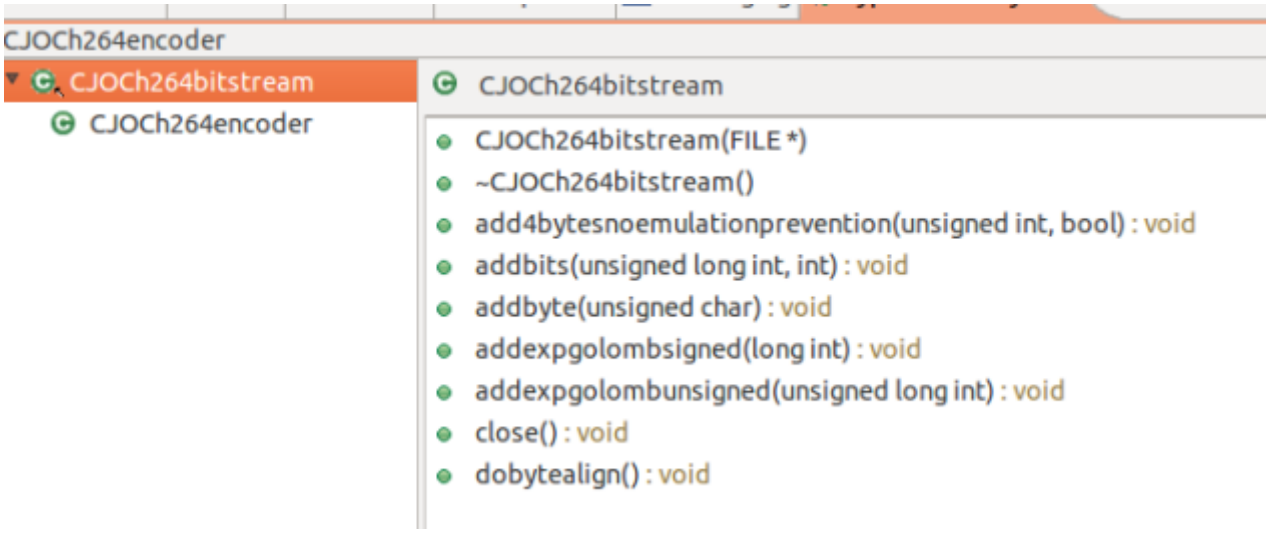
```
try
{
        //Instantiate the h264 coder
        CJOCh264encoder *ph264encoder = new CJOCh264encoder(pf

        //Initialize the h264 coder with frame parameters
        ph264encoder->IniCoder(nImWidth,nImHeight,nFps,CJOCh26

        int nSavedFrames = 0;
        char szLog[256];

        //Iterate trough all frames
        while (! feof(pfsrc))
        {
                //Get frame pointer to fill
                void *pFrame = ph264encoder->GetFramePtr ();

                //Get the size allocated in pFrame
                unsigned int nFrameSize = ph264encoder->GetFra

                //Load frame from disk and load it into pFrame
                size_t nreaded = fread (pFrame,1, nFrameSize,
                if (nreaded != nFrameSize)
                {
                        if (! feof(pfsrc))
                                throw "Error: Reading frame";
                }
                else
                {
                        //Encode & save frame
                        ph264encoder->CodeAndSaveFrame();

                        //Get the number of saved frames
                        nSavedFrames = ph264encoder->GetSavedF

                        //Show the number of saved / encoded f
                        sprintf(szLog,"Saved frame num: %d", n
                        puts (szLog);
                }
        }

        //Close encoder
        ph264encoder->CloseCoder();

        //Set return code to 0
        nRc = 0;
}
catch (const char *szErrorDesc)
{
        //Show the error description on console
        puts (szErrorDesc);
```

```
        }

        //Close previously opened files
        if (pfsrc != NULL)
                fclose (pfsrc);

        if (pfdst != NULL)
                fclose (pfdst);

        return nRc;
}
```

Classes

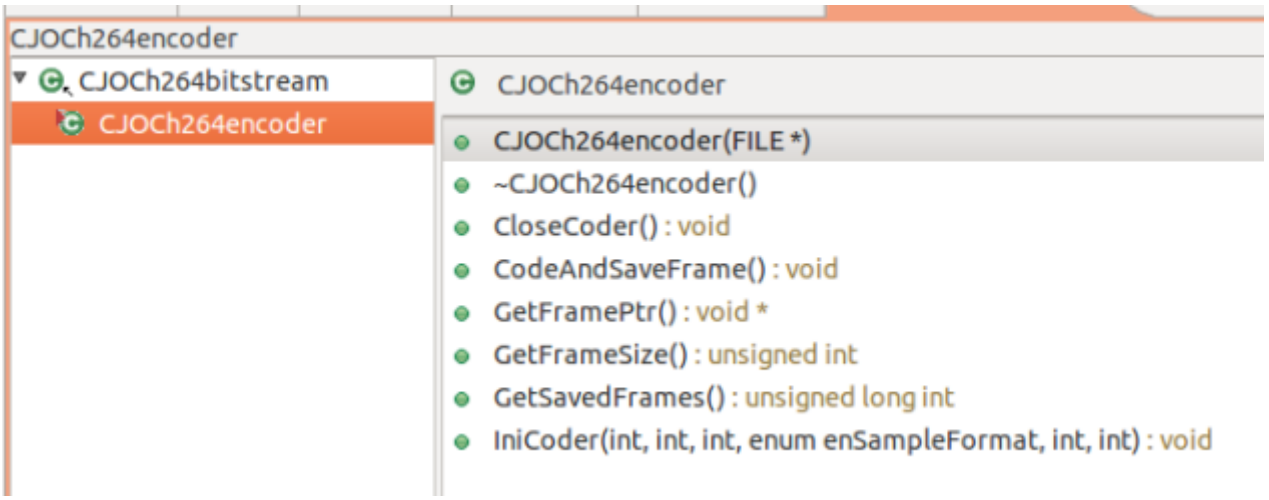
The implementation of this h264 minimal encoder it is based in two classes:

- **CJOCh264bitstream**
  - It contains useful functions to create the bit oriented stream, it has an exp Golomb coder as well.
- **CJOCh264encoder** : CJOCh264bitstream
  - It derives from CJOCh264Bitstream and it contains the h264 oriented functions.



(https://jordicenzano.files.wordpress.com/2014/08/h264bitstream.png)

Figure 2: The CJOCh264bitstream class with its public functions



(https://jordicenzano.files.wordpress.com/2014/08/h264encoder.png)

Figure 3: The CJOCh264encoder class with its public functions

Source code

- I have tried to make a readable code including comments and keeping a logical order of functions.
- You can see and download the latest version of the source code of this “experiment” from this github link: h264simpleCoder (https://github.com/jordicenzano/h264simpleCoder)
  - You will find the following files: h264simpleCoder.cpp, CJOCh264bitstream.h, CJOCh264bitstream.cpp, CJOCh264encoder
- In the following sections you can see the source code of CJOCh264bitstream and CJOCh264encoder classes

CJOCh264bitstream (.h and .cpp)

```
/*
 * CJOCh264bitstream.h
 *
 *  Created on: Aug 23, 2014
 *      Author: Jordi Cenzano (www.jordicenzano.name)
 */

#ifndef CJOCH264BITSTREAM_H_
#define CJOCH264BITSTREAM_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

//! h264 bitstream class
/*!
 It is used to create the h264 bit oriented stream, it contains differ
 */
class CJOCh264bitstream
{
#define BUFFER_SIZE_BITS 24                     /*! Buffer size in bit
#define BUFFER_SIZE_BYTES (24/8)        /*! Buffer size in bytes used

#define H264_EMULATION_PREVENTION_BYTE 0x03             /*! Emulation

private:

        /*! Buffer  */
        unsigned char m_buffer[BUFFER_SIZE_BITS];

        /*! Bit buffer index  */
        unsigned int m_nLastbitinbuffer;

        /*! Starting byte indicator  */
        unsigned int m_nStartingbyte;

        /*! Pointer to output file */
        FILE *m_pOutFile;

        //! Clears the buffer
        void clearbuffer();

        //! Returns the nNumbit value (1 or 0) of lval
        /*!
                \param lval number to extract the nNumbit value
                \param nNumbit Bit position that we want to know if i
                \return bit value (1 or 0)
         */
        static int getbitnum (unsigned long lval, int nNumbit);
```

```
        //! Adds 1 bit to the end of h264 bitstream
        /*!
                \param nVal bit to add at the end of h264 bitstream
         */
        void addbittostream (int nVal);

        //! Adds 8 bit to the end of h264 bitstream (it is optimized f
        /*!
                \param nVal byte to add at the end of h264 bitstream
         */
        void addbytetostream (int nVal);

        //! Save all buffer to file
        /*!
                \param bemulationprevention Indicates if it will inse
         */
        void savebufferbyte(bool bemulationprevention = true);

public:
        //! Constructor
        /*!
                \param pOutBinaryFile The output file pointer
         */
        CJOCh264bitstream(FILE *pOutBinaryFile);

        //! Destructor
        virtual ~CJOCh264bitstream();

        //! Add 4 bytes to h264 bistream without taking into acount th
        /*!
                \param nVal The 32b value to add
                \param bDoAlign Indicates if the function will insert
         */
        void add4bytesnoemulationprevention (unsigned int nVal, bool b

        //! Adds nNumbits of lval to the end of h264 bitstream
        /*!
                \param nVal value to add at the end of the h264 strea
                \param nNumbits number of bits of lval that will be a
         */
        void addbits (unsigned long lval, int nNumbits);

        //! Adds lval to the end of h264 bitstream using exp golomb co
        /*!
                \param nVal value to add at the end of the h264 strea
         */
        void addexpgolombunsigned (unsigned long lval);

        //! Adds lval to the end of h264 bitstream using exp golomb co
        /*!
                \param nVal value to add at the end of the h264 strea
         */
```

```
        void addexpgolombsigned (long lval);

        //! Adds 0 to the end of h264 bistream in order to leave a byt
        void dobytealign();

        //! Adds cByte (8 bits) to the end of h264 bitstream. This fun
        /*!
                \param cByte value to add at the end of the h264 stre
         */
        void addbyte (unsigned char cByte);

        //! Close the h264 stream saving to disk the last remaing bits
        void close();
};

#endif /* CJOCH264BITSTREAM_H_ */
```

```
/*
 * CJOCh264bitstream.cpp
 *
 *  Created on: Aug 23, 2014
 *      Author: Jordi Cenzano (www.jordicenzano.name)
 */

#include "CJOCh264bitstream.h"

CJOCh264bitstream::CJOCh264bitstream(FILE *pOutBinaryFile)
{
        clearbuffer();

        m_pOutFile = pOutBinaryFile;
}

CJOCh264bitstream::~CJOCh264bitstream()
{
        close();
}

void CJOCh264bitstream::clearbuffer()
{
        memset (&m_buffer,0,sizeof (unsigned char)* BUFFER_SIZE_BITS);
        m_nLastbitinbuffer = 0;
        m_nStartingbyte = 0;
}

int CJOCh264bitstream::getbitnum (unsigned long lval, int nNumbit)
{
        int lrc = 0;

        unsigned long lmask = (unsigned long) pow((unsigned long)2,(un
        if ((lval & lmask) > 0)
                lrc = 1;

        return lrc;
}

void CJOCh264bitstream::addbittostream (int nVal)
{
        if (m_nLastbitinbuffer >= BUFFER_SIZE_BITS)
        {
                //Must be aligned, no need to do dobytealign();
                savebufferbyte();
        }

        //Use circular buffer of BUFFER_SIZE_BYTES
        int nBytePos = (m_nStartingbyte + (m_nLastbitinbuffer / 8)) %
        //The first bit to add is on the left
```

```
        int nBitPosInByte = 7 - m_nLastbitinbuffer % 8;

        //Get the byte value from buffer
        int nValTmp = m_buffer[nBytePos];

        //Change the bit
        if (nVal > 0)
                nValTmp = (nValTmp | (int) pow(2,nBitPosInByte));
        else
                nValTmp = (nValTmp & ~((int) pow(2,nBitPosInByte)));

        //Save the new byte value to the buffer
        m_buffer[nBytePos] = (unsigned char) nValTmp;

        m_nLastbitinbuffer++;
}

void CJOCh264bitstream::addbytetostream (int nVal)
{
        if (m_nLastbitinbuffer >= BUFFER_SIZE_BITS)
        {
                //Must be aligned, no need to do dobytealign();
                savebufferbyte();
        }

        //Used circular buffer of BUFFER_SIZE_BYTES
        int nBytePos = (m_nStartingbyte + (m_nLastbitinbuffer / 8)) %
        //The first bit to add is on the left
        int nBitPosInByte = 7 - m_nLastbitinbuffer % 8;

        //Check if it is byte aligned
        if (nBitPosInByte != 7)
                throw "Error: inserting not aligment byte";

        //Add all byte to buffer
        m_buffer[nBytePos] = (unsigned char) nVal;

        m_nLastbitinbuffer = m_nLastbitinbuffer + 8;
}

void CJOCh264bitstream::dobytealign()
{
        //Check if the last bit in buffer is multiple of 8
        int nr = m_nLastbitinbuffer % 8;
        if ((nr % 8) != 0)
                m_nLastbitinbuffer = m_nLastbitinbuffer + (8 - nr);
}

void CJOCh264bitstream::savebufferbyte(bool bemulationprevention)
{
        bool bemulationpreventionexecuted = false;
```

```
        if (m_pOutFile == NULL)
                throw "Error: out file is NULL";

        //Check if the last bit in buffer is multiple of 8
        if ((m_nLastbitinbuffer % 8)  != 0)
                throw "Error: Save to file must be byte aligned";

        if ((m_nLastbitinbuffer / 8) <= 0)
                throw "Error: NO bytes to save";

        if (bemulationprevention == true)
        {
                //Emulation prevention will be used:
                /*As per h.264 spec,
                rbsp_data shouldn't contain
                                - 0x 00 00 00
                                - 0x 00 00 01
                                - 0x 00 00 02
                                - 0x 00 00 03

                rbsp_data shall be in the following way
                                - 0x 00 00 03 00
                                - 0x 00 00 03 01
                                - 0x 00 00 03 02
                                - 0x 00 00 03 03
                */

                //Check if emulation prevention is needed (emulation p
                if ((m_buffer[((m_nStartingbyte + 0) % BUFFER_SIZE_BYT
                {
                        int nbuffersaved = 0;
                        unsigned char cEmulationPreventionByte = H264_

                        //Save 1st byte
                        fwrite(&m_buffer[((m_nStartingbyte + nbuffersa
                        nbuffersaved ++;

                        //Save 2st byte
                        fwrite(&m_buffer[((m_nStartingbyte + nbuffersa
                        nbuffersaved ++;

                        //Save emulation prevention byte
                        fwrite(&cEmulationPreventionByte, 1, 1, m_pOut

                        //Save the rest of bytes (usually 1)
                        while (nbuffersaved < BUFFER_SIZE_BYTES)
                        {
                                fwrite(&m_buffer[((m_nStartingbyte + n
                                nbuffersaved++;
                        }

                        //All bytes in buffer are saved, so clear the
```

```
                        clearbuffer();

                        bemulationpreventionexecuted = true;
                }
        }

        if (bemulationpreventionexecuted == false)
        {
                //No emulation prevention was used

                //Save the oldest byte in buffer
                fwrite(&m_buffer[m_nStartingbyte], 1, 1, m_pOutFile);

                //Move the index
                m_buffer[m_nStartingbyte] = 0;
                m_nStartingbyte++;
                m_nStartingbyte = m_nStartingbyte % BUFFER_SIZE_BYTES;
                m_nLastbitinbuffer = m_nLastbitinbuffer - 8;
        }
}

//Public functions

void CJOCh264bitstream::addbits (unsigned long lval, int nNumbits)
{
        if ((nNumbits <= 0 )||(nNumbits > 64))
                throw "Error: numbits must be between 1 ... 64";

        int nBit = 0;
        int n = nNumbits-1;
        while (n >= 0)
        {
                nBit = getbitnum (lval,n);
                n--;

                addbittostream (nBit);
        }
}

void CJOCh264bitstream::addbyte (unsigned char cByte)
{
        //Byte alignment optimization
        if ((m_nLastbitinbuffer % 8)  == 0)
        {
                addbytetostream (cByte);
        }
        else
        {
                addbits (cByte, 8);
        }
}
```

```
void CJOCh264bitstream::addexpgolombunsigned (unsigned long lval)
{
        //it implements unsigned exp golomb coding

        unsigned long lvalint = lval + 1;
        int nnumbits = log2 (lvalint) + 1;

        for (int n = 0; n < (nnumbits-1); n++)
                addbits(0,1);

        addbits(lvalint,nnumbits);
}

void CJOCh264bitstream::addexpgolombsigned (long lval)
{
        //it implements a signed exp golomb coding

        unsigned long lvalint = abs(lval) * 2 - 1;
        if (lval <= 0)
                lvalint =  2 * abs(lval);

        addexpgolombunsigned(lvalint);
}

void CJOCh264bitstream::add4bytesnoemulationprevention (unsigned int n
{
        //Used to add NAL header stream
        //Remember: NAL header is byte oriented
        if (bDoAlign == true)
                dobytealign();

        if ((m_nLastbitinbuffer % 8) != 0)
                throw "Error: Save to file must be byte aligned";

        while (m_nLastbitinbuffer != 0)
                savebufferbyte();

        unsigned char cbyte = (nVal & 0xFF000000)>>24;
        fwrite(&cbyte, 1, 1, m_pOutFile);

        cbyte = (nVal & 0x00FF0000)>>16;
        fwrite(&cbyte, 1, 1, m_pOutFile);

        cbyte = (nVal & 0x0000FF00)>>8;
        fwrite(&cbyte, 1, 1, m_pOutFile);

        cbyte = (nVal & 0x000000FF);
        fwrite(&cbyte, 1, 1, m_pOutFile);
}

void CJOCh264bitstream::close()
{
```

```
        //Flush the data in stream buffer

        dobytealign();

        while (m_nLastbitinbuffer != 0)
                savebufferbyte();
}
```

CJOCh264encoder (.h and .cpp)

```
/*
 * CJOCh264encoder.h
 *
 *  Created on: Aug 17, 2014
 *      Author: Jordi Cenzano (www.jordicenzano.name)
 */

#ifndef CJOCH264ENCODER_H_
#define CJOCH264ENCODER_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "CJOCh264bitstream.h"

//! h264 encoder class
/*!
 It is used to create the h264 compliant stream
 */
class CJOCh264encoder : CJOCh264bitstream
{
public:

        /**
         * Allowed sample formats
         */
        enum enSampleFormat
        {
                SAMPLE_FORMAT_YUV420p//!< SAMPLE_FORMAT_YUV420p
        };

private:
        /*!Set the used Y macroblock size for I PCM in YUV420p */
        #define MACROBLOCK_Y_WIDTH      16
        #define MACROBLOCK_Y_HEIGHT     16

        /*!Set time base in Hz */
        #define TIME_SCALE_IN_HZ        27000000

        /*!Pointer to pixels */
        typedef struct
        {
                unsigned char *pYCbCr;
        }YUV420p_frame_t;

        /*! Frame  */
        typedef struct
        {
                enSampleFormat sampleformat;    /*!< Sample format */
```

```
        unsigned int nYwidth;                   /*!< Y (lumina
        unsigned int nYheight;                  /*!< Y (lumina
        unsigned int nCwidth;                   /*!< C (Cromin
        unsigned int nCheight;                  /*!< C (Cromin

        unsigned int nYmbwidth;                 /*!< Y (lumina
        unsigned int nYmbheight;                /*!< Y (lumina
        unsigned int nCmbwidth;                 /*!< Y (Cromin
        unsigned int nCmbheight;                /*!< Y (Cromin

        YUV420p_frame_t yuv420pframe;   /*!< Pointer to curren
        unsigned int nyuv420pframesize; /*!< Size in bytes of
}frame_t;

/*! The frame var*/
frame_t m_frame;

/*! The frames per second var*/
int m_nFps;

/*! Number of frames sent to the output */
unsigned long m_lNumFramesAdded;

//! Frees the frame yuv420pframe allocated memory
void free_video_src_frame ();

//! Allocs the frame yuv420pframe memory according to the fram
void alloc_video_src_frame ();

//! Creates SPS NAL and add it to the output
/*!
        \param nImW Frame width in pixels
        \param nImH Frame height in pixels
        \param nMbW macroblock width in pixels
        \param nMbH macroblock height in pixels
        \param nFps frames x second (tipical values are: 25, 3
        \param nSARw Indicates the horizontal size of the samp
        \param nSARh Indicates the vertical size of the sample
 */
void create_sps (int nImW, int nImH, int nMbW, int nMbH, int n

//! Creates PPS NAL and add it to the output
void create_pps ();

//! Creates Slice NAL and add it to the output
/*!
        \param lFrameNum number of frame
 */
void create_slice_header(unsigned long lFrameNum);

//! Creates macroblock header and add it to the output
void create_macroblock_header ();
```

```
        //! Creates the slice footer and add it to the output
        void create_slice_footer();

        //! Creates SPS NAL and add it to the output
        /*!
                \param nYpos First vertical macroblock pixel inside th
                \param nYpos nXpos horizontal macroblock pixel inside
         */
        void create_macroblock(unsigned int nYpos, unsigned int nXpos)

public:
        //! Constructor
        /*!
                 \param pOutFile The output file pointer
         */
        CJOCh264encoder(FILE *pOutFile);

        //! Destructor
        virtual ~CJOCh264encoder();

        //! Initializes the coder
        /*!
                \param nImW Frame width in pixels
                \param nImH Frame height in pixels
                \param nFps Desired frames per second of the output fi
                \param SampleFormat Sample format if the input file. I
                \param nSARw Indicates the horizontal size of the samp
                \param nSARh Indicates the vertical size of the sample
        */
        void IniCoder (int nImW, int nImH, int nImFps, CJOCh264encoder

        //! Returns the frame pointer
        /*!
                \return Frame pointer ready to fill with frame pixels
        */
        void* GetFramePtr();

        //! Returns the allocated frame memory in bytes
        /*!
                \return The allocated memory to store the frame data
        */
        unsigned int GetFrameSize();

        //! It codes the frame that is in frame memory a it saves the
        void CodeAndSaveFrame();

        //! Returns number of coded frames
        /*!
                \return The number of coded frames
        */
        unsigned long GetSavedFrames();
```

```
        //! Flush all data and save the trailing bits
        void CloseCoder ();
};

#endif /* CJOCH264ENCODER_H_ */
```

```
/*
 * CJOCh264encoder.cpp
 *
 *  Created on: Aug 17, 2014
 *      Author: Jordi Cenzano (www.jordicenzano.name)
 */

#include "CJOCh264encoder.h"

//Private functions

//Contructor
CJOCh264encoder::CJOCh264encoder(FILE *pOutFile):CJOCh264bitstream(pOu
{
        m_lNumFramesAdded = 0;

        memset (&m_frame, 0, sizeof (frame_t));
        m_nFps = 25;
}

//Destructor
CJOCh264encoder::~CJOCh264encoder()
{
        free_video_src_frame ();
}

//Free the allocated video frame mem
void CJOCh264encoder::free_video_src_frame ()
{
        if (m_frame.yuv420pframe.pYCbCr != NULL)
                free (m_frame.yuv420pframe.pYCbCr);

        memset (&m_frame, 0, sizeof (frame_t));
}

//Alloc mem to store a video frame
void CJOCh264encoder::alloc_video_src_frame ()
{
        if (m_frame.yuv420pframe.pYCbCr != NULL)
                throw "Error: null values in frame";

        int nYsize = m_frame.nYwidth * m_frame.nYheight;
        int nCsize = m_frame.nCwidth * m_frame.nCheight;
        m_frame.nyuv420pframesize = nYsize + nCsize + nCsize;

        m_frame.yuv420pframe.pYCbCr = (unsigned char*) malloc (sizeof

        if (m_frame.yuv420pframe.pYCbCr == NULL)
                throw "Error: memory alloc";
}
```

```
//Creates and saves the NAL SPS (including VUI) (one per file)
void CJOCh264encoder::create_sps (int nImW, int nImH, int nMbW, int nM
{
        add4bytesnoemulationprevention (0x000001); // NAL header
        addbits (0x0,1); // forbidden_bit
        addbits (0x3,2); // nal_ref_idc
        addbits (0x7,5); // nal_unit_type : 7 ( SPS )
        addbits (0x42,8); // profile_idc = baseline ( 0x42 )
        addbits (0x0,1); // constraint_set0_flag
        addbits (0x0,1); // constraint_set1_flag
        addbits (0x0,1); // constraint_set2_flag
        addbits (0x0,1); // constraint_set3_flag
        addbits (0x0,1); // constraint_set4_flag
        addbits (0x0,1); // constraint_set5_flag
        addbits (0x0,2); // reserved_zero_2bits /* equal to 0 */
        addbits (0x0a,8); // level_idc: 3.1 (0x0a)
        addexpgolombunsigned(0); // seq_parameter_set_id
        addexpgolombunsigned(0); // log2_max_frame_num_minus4
        addexpgolombunsigned(0); // pic_order_cnt_type
        addexpgolombunsigned(0); // log2_max_pic_order_cnt_lsb_minus4
        addexpgolombunsigned(0); // max_num_refs_frames
        addbits(0x0,1); // gaps_in_frame_num_value_allowed_flag

        int nWinMbs = nImW / nMbW;
        addexpgolombunsigned(nWinMbs-1); // pic_width_in_mbs_minus_1
        int nHinMbs = nImH / nMbH;
        addexpgolombunsigned(nHinMbs-1); // pic_height_in_map_units_mi

        addbits(0x1,1); // frame_mbs_only_flag
        addbits(0x0,1); // direct_8x8_interfernce
        addbits(0x0,1); // frame_cropping_flag
        addbits(0x1,1); // vui_parameter_present

        //VUI parameters (AR, timming)
        addbits(0x1,1); //aspect_ratio_info_present_flag
        addbits(0xFF,8); //aspect_ratio_idc = Extended_SAR

        //AR
        addbits(nSARw, 16); //sar_width
        addbits(nSARh, 16); //sar_height

        addbits(0x0,1); //overscan_info_present_flag
        addbits(0x0,1); //video_signal_type_present_flag
        addbits(0x0,1); //chroma_loc_info_present_flag
        addbits(0x1,1); //timing_info_present_flag

        unsigned int nnum_units_in_tick = TIME_SCALE_IN_HZ / (2*nFps);
        addbits(nnum_units_in_tick,32); //num_units_in_tick
        addbits(TIME_SCALE_IN_HZ,32); //time_scale
        addbits(0x1,1);  //fixed_frame_rate_flag
```

```
        addbits(0x0,1);  //nal_hrd_parameters_present_flag
        addbits(0x0,1);  //vcl_hrd_parameters_present_flag
        addbits(0x0,1);  //pic_struct_present_flag
        addbits(0x0,1);  //bitstream_restriction_flag
        //END VUI

        addbits(0x0,1); // frame_mbs_only_flag
        addbits(0x1,1); // rbsp stop bit

        dobytealign();
}

//Creates and saves the NAL PPS (one per file)
void CJOCh264encoder::create_pps ()
{
        add4bytesnoemulationprevention (0x000001); // NAL header
        addbits (0x0,1); // forbidden_bit
        addbits (0x3,2); // nal_ref_idc
        addbits (0x8,5); // nal_unit_type : 8 ( PPS )
        addexpgolombunsigned(0); // pic_parameter_set_id
        addexpgolombunsigned(0); // seq_parameter_set_id
        addbits (0x0,1); // entropy_coding_mode_flag
        addbits (0x0,1); // bottom_field_pic_order_in frame_present_fl
        addexpgolombunsigned(0); // nun_slices_groups_minus1
        addexpgolombunsigned(0); // num_ref_idx10_default_active_minus
        addexpgolombunsigned(0); // num_ref_idx11_default_active_minus
        addbits (0x0,1); // weighted_pred_flag
        addbits (0x0,2); // weighted_bipred_idc
        addexpgolombsigned(0); // pic_init_qp_minus26
        addexpgolombsigned(0); // pic_init_qs_minus26
        addexpgolombsigned(0); // chroma_qp_index_offset
        addbits (0x0,1); //deblocking_filter_present_flag
        addbits (0x0,1); // constrained_intra_pred_flag
        addbits (0x0,1); //redundant_pic_ent_present_flag
        addbits(0x1,1); // rbsp stop bit

        dobytealign();
}

//Creates and saves the NAL SLICE (one per frame)
void CJOCh264encoder::create_slice_header(unsigned long lFrameNum)
{
        add4bytesnoemulationprevention (0x000001); // NAL header
        addbits (0x0,1); // forbidden_bit
        addbits (0x3,2); // nal_ref_idc
        addbits (0x5,5); // nal_unit_type : 5 ( Coded slice of an IDR
        addexpgolombunsigned(0); // first_mb_in_slice
        addexpgolombunsigned(7); // slice_type
        addexpgolombunsigned(0); // pic_param_set_id

        unsigned char cFrameNum = lFrameNum % 16; //(2⁴)
        addbits (cFrameNum,4); // frame_num ( numbits = v = log2_max_f
```

```
        unsigned long lidr_pic_id = lFrameNum % 512;
        //idr_pic_flag = 1
        addexpgolombunsigned(lidr_pic_id); // idr_pic_id
        addbits(0x0,4); // pic_order_cnt_lsb (numbits = v = log2_max_f
        addbits(0x0,1); //no_output_of_prior_pics_flag
        addbits(0x0,1); //long_term_reference_flag
        addexpgolombsigned(0); //slice_qp_delta

        //Probably NOT byte aligned!!!
}

//Creates and saves the SLICE footer (one per SLICE)
void CJOCh264encoder::create_slice_footer()
{
        addbits(0x1,1); // rbsp stop bit
}

//Creates and saves the macroblock header(one per macroblock)
void CJOCh264encoder::create_macroblock_header ()
{
        addexpgolombunsigned(25); // mb_type (I_PCM)
}

//Creates & saves a macroblock (coded INTRA 16x16)
void CJOCh264encoder::create_macroblock(unsigned int nYpos, unsigned i
{
        unsigned int x,y;

        create_macroblock_header();

        dobytealign();

        //Y
        unsigned int nYsize = m_frame.nYwidth * m_frame.nYheight;
        for(y = nYpos * m_frame.nYmbheight; y < (nYpos+1) * m_frame.nY
        {
                for (x = nXpos * m_frame.nYmbwidth; x < (nXpos+1) * m_
                {
                        addbyte (m_frame.yuv420pframe.pYCbCr[(y * m_fr
                }
        }

        //Cb
        unsigned int nCsize = m_frame.nCwidth * m_frame.nCheight;
        for(y = nYpos * m_frame.nCmbheight; y < (nYpos+1) * m_frame.nC
        {
                for (x = nXpos * m_frame.nCmbwidth; x < (nXpos+1) * m_
                {
                        addbyte(m_frame.yuv420pframe.pYCbCr[nYsize + (
                }
        }
```

```
        //Cr
        for(y = nYpos * m_frame.nCmbheight; y < (nYpos+1) * m_frame.nC
        {
                for (x = nXpos * m_frame.nCmbwidth; x < (nXpos+1) * m_
                {
                        addbyte(m_frame.yuv420pframe.pYCbCr[nYsize + n
                }
        }
}

//public functions

//Initilizes the h264 coder (mini-coder)
void CJOCh264encoder::IniCoder (int nImW, int nImH, int nImFps, CJOCh2
{
        m_lNumFramesAdded = 0;
        if (SampleFormat != SAMPLE_FORMAT_YUV420p)
                throw "Error: SAMPLE FORMAT not allowed. Only yuv420p

        free_video_src_frame ();

        //Ini vars
        m_frame.sampleformat = SampleFormat;
        m_frame.nYwidth = nImW;
        m_frame.nYheight = nImH;
        if (SampleFormat == SAMPLE_FORMAT_YUV420p)
        {
                //Set macroblock Y size
                m_frame.nYmbwidth = MACROBLOCK_Y_WIDTH;
                m_frame.nYmbheight = MACROBLOCK_Y_HEIGHT;
                //Set macroblock C size (in YUV420 is 1/2 of Y)
                m_frame.nCmbwidth = MACROBLOCK_Y_WIDTH/2;
                m_frame.nCmbheight = MACROBLOCK_Y_HEIGHT/2;
                //Set C size
                m_frame.nCwidth = m_frame.nYwidth / 2;
                m_frame.nCheight = m_frame.nYheight / 2;
                //In this implementation only picture sizes multiples
                if (((nImW % MACROBLOCK_Y_WIDTH) != 0)||((nImH % MACRO
                        throw "Error: size not allowed. Only multiples
        }

        m_nFps = nImFps;

        //Alloc mem for 1 frame
        alloc_video_src_frame ();

        //Create h264 SPS & PPS
        create_sps (m_frame.nYwidth , m_frame.nYheight, m_frame.nYmbwi
        create_pps ();
}
```

```
//Returns the frame pointer to load the video frame
void* CJOCh264encoder::GetFramePtr()
{
        if (m_frame.yuv420pframe.pYCbCr == NULL)
                throw "Error: video frame is null (not initialized)";

        return (void*) m_frame.yuv420pframe.pYCbCr;
}

//Returns the the allocated size for video frame
unsigned int CJOCh264encoder::GetFrameSize()
{
        return m_frame.nyuv420pframesize;
}

//Codifies & save the video frame (it only uses 16x16 intra PCM -> NO
void CJOCh264encoder::CodeAndSaveFrame()
{
        //The slice header is not byte aligned, so the first macrobloc
        create_slice_header (m_lNumFramesAdded);

        //Loop over macroblock size
        unsigned int y,x;
        for (y = 0; y < m_frame.nYheight / m_frame.nYmbheight; y++)
        {
                for (x = 0; x < m_frame.nYwidth / m_frame.nYmbwidth; x
                {
                        create_macroblock(y, x);
                }
        }

        create_slice_footer();
        dobytealign();

        m_lNumFramesAdded++;
}

//Returns the number of codified frames
unsigned long CJOCh264encoder::GetSavedFrames()
{
        return m_lNumFramesAdded;
}

//Closes the h264 coder saving the last bits in the buffer
void CJOCh264encoder::CloseCoder ()
{
        close();
}
```

Future work

- Evolve the code implementing h264 intra frame compression techniques, such as intra prediction, CAVLC, etc.
- Implement h264 inter frame compression techniques, such block matching

FILED UNDER PAPERS & ARTICLES TAGGED WITH C++, GOLOMB, H264, MACROBLOCK, PPS, SLICE, SOURCE CODE, SPS, VIDEO, VIDEO COFING

6 Responses to *A minimal h264 “encoder” (C++)*

Nabeel says:
06/02/2015 at 14:57
Hello, Could you please help me for encoding and decoding the life time stream of video capturing from webcam? or let us CCTV camera.

Reply

Jordi Cenzano says:
09/02/2015 at 01:12
I’m sorry, I’m so busy right now. Try with ffmpeg. It is a good tool to start

Reply

Roger Hardiman says:
28/03/2015 at 02:17
Thanks for posting your project. This is a nice step up from the original tiny encoder and it is good to see the sps and pps creation code.
Roger

Reply

m4c0 says:
16/10/2016 at 10:37
What about using GitHub for this code? It’s quite hard to follow it on a WP site…

Reply

Jordi Cenzano says:
22/10/2016 at 12:52
Thanks for your feedback. I agree with you that github is a better place for code. My latest experiments are there.
Will try to put this one there too, but I can not commit to an ETA

Reply

Roger Hardiman says:
11/09/2017 at 23:50
https://github.com/jordicenzano/h264simpleCoder

Reply

Blog at WordPress.com.