libr263 video compression library

Roalt Aalmoes

draft of July 10, 2004

Abstract

This paper describes the usage of the libr263 library, a fast implementation of an H.263 encoder. The implementations does not feature the advantaged options, as they are currently too computational expensive for real-time compression. However, the encoder includes variable-sized frames encoding and an implementation of a logarithmic search algorithm. The encoder itself is optimized for 64-bit processors, although it also works on 32-bit processors. The encoder has the ability to encode only parts of the picture by supplying an array of booleans for each macroblock.

1 Introduction

The implementation of the encoder is derived from the telenor TMN 1.6 implementation. Where the telenor implementation is focussed on functionality of all (advanced) options, this implementation is aimed at real-time compression. This results in a slim implementation with no *processing time*-consuming advanced features. The most significant difference with an H.261 encoder is the use of half-pixel prediction which reduces bitrate significantly.

This library function is currently in beta-testing. Please feel free to send any comments to aalmoes@huygens.nl. There is not a decoder, as the TMN 1.7 implementation from telenor is fast and useful.

I would especially like to thank Karl Lillevold for making the original source freely available.

2 How to use the encoder

2.1 Initialization

To use the encoder, a number of variables must be declared:

```
/* Variables to be declared */
CParam cparams;
Bits bits;
```

The cparams variable will hold parameter information for the encoder. The `CParam` structure is shown below:

```
typedef struct compression_parameters {
/* Contains all the parameters that are needed for
   encoding plus all the status between two encodings */
  int half_pixel_searchwindow; /* size of search window in half pixels
                                  if this value is 0, no search is performed
                               */
  int format;
  int pels;                    /* Only used when format == CPARAM_OTHER */
  int lines;                   /* Only used when format == CPARAM_OTHER */
  int inter;                   /* TRUE of INTER frame encoded frames,
                                  FALSE for INTRA frames */
  int search_method;           /* DEF_EXHAUSTIVE or DEF_LOGARITHMIC */
  int advanced_method;         /* TRUE : Use array to determine
                                          macroblocks in INTER frame
                                          mode to be encoded */
  int Q_inter;                 /* Quantization factor for INTER frames */
  int Q_intra;                 /* Quantization factor for INTRA frames */
  unsinged int *data;             /* source data  */
  unsigned int *interpolated_lum;  /* intepolated reconstructed
                                      luminance part (internal) */
  unsigned int *recon;            /* Reconstructed copy of compressed frame */
  int *EncodeThisBlock;
                               /* Array of mbr*mbc when advanced_method
                                  is used */
} CParam;
```

The following constants are also defined:

```
/* Compression parameter structure */

#define CPARAM_INTER TRUE
#define CPARAM_INTRA FALSE
#define CPARAM_EXHAUSTIVE TRUE
#define CPARAM_LOGARITHMIC FALSE
#define CPARAM_ADVANCED TRUE
#define CPARAM_NOADVANCED FALSE

#define CPARAM_QCIF 0
#define CPARAM_CIF 1
#define CPARAM_4CIF 2
#define CPARAM_16CIF 3
#define CPARAM_SQCIF 4
```

```
#define CPARAM_OTHER 99

#define CPARAM_DEFAULT_INTER_Q 8
#define CPARAM_DEFAULT_INTRA_Q 8
#define CPARAM_DEFAULT_SEARCHWINDOW 3
#define CPARAM_DEFAULT_INTER = CPARAM_INTRA
#define CPARAM_DEFAULT_SEARCH_METHOD = CPARAM_LOGARITHMIC
#define CPARAM_DEFAULT_ADVANCED_METHOD = CPARAM_NOADVANCED
```

The CPARAM_DEFAULT_ defines give the default values after initialization. The initialization of the cparams structure is done as follows:

```
/* Initialisation */
cparams.format = CPARAM_QCIF; /* For quarter-CIF sized frames */
InitCompress(&cparams);  /* Use standard compression parameters */
WriteByteFunction = OwnWriteFunction;
```

The first assignment determines the size of the video frames that are compressed. To use the 176×144 QCIF format, assign here `CPARAM_QCIF` to `format`. To use the 352×288 CIF format, define here `CPARAM_CIF` to `format`. Other format that may be used are `CPARAM_SQCIF`, `CPARAM_4CIF` and `CPARAM_16CIF` (see the file `libr263.h`). To use an alternative format, assign `CPARAM_OTHER` to `format` and also assign the pixels per line to `pels` and the number of lines to `lines`. In any format, the size of the two chrominance components is half the lines and pels size. For example, the QCIF format has a chrominance size of 88×72, which means that the total number of integers a frame consists of is $176 \times 144 + 88 \times 72 + 88 \times 72 = 38016$.

The `InitCompress` function allocates some local memory and initializes some structures. It also fills in the defaults values in the cparam structure. The only value that must be initialized *before* the `InitCompress` call is `format`. Beware that you overrule the default values *a*fter the InitCompress function. The WriteByteFunction has the following type:

```
typedef void (*WriteByte) (int);

/* Global variable */
WriteByte WriteByteFunction;
```

In order to use the output data of the compression, you should assign this value to your own function with the same type. An example WriteByteFunction is given below:

```
void OwnWriteFunction(int byte)
{
  putc(byte, outputstream);

  return;
}
```

`byte` is the byte to be written and `outputstream` is a file pointer defined by yourself.

2.2 Intra encoding

After these function are defined, a frame can be INTRA encoded. Remember that the first frame must always be an INTRA frame:

```
  /* Parameters to encode INTRA */
  cparams.inter = CPARAM_INTRA;
  cparams.data = (unsinged int *) &qcif_frame;
  /* struct qcif qcif_frame holds QCIF frame */
  CompressToH263(&cparams, &bits);
```

The `data` field holds the frame. The structure of this frame for the QCIF format is as follows:

```
#define QCIF_YWIDTH 176
#define QCIF_YHEIGHT 144
#define QCIF_UWIDTH 88
#define QCIF_UHEIGHT 72
#define QCIF_VWIDTH 88
#define QCIF_VHEIGHT 72

struct qcif {
  unsigned int Y[QCIF_YHEIGHT][QCIF_YWIDTH];
  unsigned int U[QCIF_UHEIGHT][QCIF_UWIDTH];
  unsigned int V[QCIF_VHEIGHT][QCIF_VWIDTH];
};
```

Other parameters that can be set for INTRA encoding is the quantization factor `Q_intra`. Other parameters do not effect intra encoding.

2.3 inter encoding

To encode a frame INTER, the following parameters must be set:

```
/* Parameters to encode INTER */
cparams.inter = CPARAM_INTER;
CompressToH263(&cparams, &bits);
```

These are the minimal parameters to be set. Other parameter that may be used are the quantization factor `Q_inter`, `half_pixel_search_window`, `advanced_method` combined with the `EncodeThisBlock` array, and the `search_method`.

2.4 Quantization setting

To set the quantization for INTRA frames, fill in a value into the `Q_intra` field. To set the quantization for INTER frames, fill in a value into the `Q_inter` field. Quantization determines directly the quality: the lower the quantization, the higher the quality but the larger the output bitstream. Although the quantization may have any values equal or higher than 1, useful values are between 4 and 16. The encoder is optimized for quantization of 8 and uses this as default.

2.5 Search method

In the `search_method` field, two available algorithms can currently be used. `CPARAM_EXHAUSTIVE` determines a full exhaustive search, while `CPARAM_LOGARITHMIC` determines a logarithmic search. Exhaustive search creates smaller bitstreams, but for large movement videos performes significant worse than logarithmic search. See also the next section.

2.6 Half pixel search window

This field gives the size of the half pixel search window, expressed in half pixel sizes from the center of the search. A value of 1 means the search area is $[-0.5..+0.5]$ in both horizontal as vertical directions. With EXHAUSTIVE search, all possible places within the area are examined, and with LOGARITHMIC, only some of them (evenly distributed across the area) are examined.

The value that may be used here is between 0 and 30, both numbers included. With EXHAUSTIVE search, using a value greater than 2 here may increase computation time significantly. With LOGARITHMIC search, performance for using larger search window areas is much better.

3 Information from the encoder

The encoder delivers information to the user in much the same way as the telenor implementation. It returns a Bits structure that gives a detailed summary of

the bits used for different parts:

```
typedef struct bits_counted {
  int Y;
  int C;
  int vec;
  int CBPY;
  int CBPCM;
  int MODB;
  int CBPB;
  int COD;
  int header;
  int DQUANT;
  int total;
  int no_inter;
  int no_inter4v;
  int no_intra;
} Bits;
```

In particularly the `total` field might be of interest. Also returned is a reconstructed picture that can be found in the `recon` field of the `CParam` structure. The encoder uses this field for Inter frame encoding, so do not alter this field unless you know what you are doing.

4 `libr263.h` include file

This section contains the `libr263.h` source file, which must be included into the source file where you use the compression library.

```
/*************************************************
 * libr263: fast H.263 encoder library
 *
 * Copyright (C) 1996, Roalt Aalmoes, Twente University
 * SPA multimedia group
 *
 * Based on Telenor TMN 1.6 encoder (Copyright (C) 1995, Telenor R&D)
 * created by Karl Lillevold
 *
 * Author encoder: Roalt Aalmoes, <aalmoes@huygens.nl>
 *
 * Date: 31-07-96
 *************************************************/

#include "rlib.h"
```

```
#ifndef LIBR263_H
#define LIBR263_H
/* This should not be changed */
#define MB_SIZE 16


/* Order of usage of lib263:
   1. Assign size of frame type to "format" field and call InitCompress()
   2. WriteByteFunction = OwnWriteFunction  (1 and 2 in arbitrary order)
   3. Set cparams and do CompressQCIFToH263(cparams) with INTRA encoding
   4. Set cparams and do CompressQCIFToH263(cparams) with either INTRA
      or INTER encoding
   5. redo 4. or to stop do 6
   6. CloseCompress()
 */

/* Compression parameter structure */

#define CPARAM_INTER TRUE
#define CPARAM_INTRA FALSE
#define CPARAM_EXHAUSTIVE TRUE
#define CPARAM_LOGARITHMIC FALSE
#define CPARAM_ADVANCED TRUE
#define CPARAM_NOADVANCED FALSE

#define CPARAM_QCIF 0
#define CPARAM_CIF 1
#define CPARAM_4CIF 2
#define CPARAM_16CIF 3
#define CPARAM_SQCIF 4
#define CPARAM_OTHER 99

#define CPARAM_DEFAULT_INTER_Q 8
#define CPARAM_DEFAULT_INTRA_Q 8
#define CPARAM_DEFAULT_SEARCHWINDOW 3

#define CPARAM_DEFAULT_INTER CPARAM_INTRA
#define CPARAM_DEFAULT_SEARCH_METHOD CPARAM_LOGARITHMIC
#define CPARAM_DEFAULT_ADVANCED_METHOD CPARAM_NOADVANCED
#define CPARAM_DEFAULT_FORMAT CPARAM_QCIF

typedef struct compression_parameters {
/* Contains all the parameters that are needed for
   encoding plus all the status between two encodings */
  int half_pixel_searchwindow; /* size of search window in half pixels
```

```
                                  if this value is 0, no search is performed
                                */
  int format;                     /*  */
  int pels;                       /* Only used when format == CPARAM_OTHER */
  int lines;                      /* Only used when format == CPARAM_OTHER */
  int inter;                      /* TRUE of INTER frame encoded frames,
                                     FALSE for INTRA frames */
  int search_method;              /* DEF_EXHAUSTIVE or DEF_LOGARITHMIC */
  int advanced_method;            /* TRUE : Use array to determine
                                             macroblocks in INTER frame
                                             mode to be encoded */
  int Q_inter;                    /* Quantization factor for INTER frames */
  int Q_intra;                    /* Quantization factor for INTRA frames */
  unsigned int *data;             /* source data in qcif format */
  unsigned int *interpolated_lum;       /* intepolated recon luminance part */
  unsigned int *recon;            /* Reconstructed copy of compressed frame */
  int *EncodeThisBlock;
                                  /* Array when advanced_method is used */
} CParam;
/* Structure for counted bits */

typedef struct bits_counted {
  int Y;
  int C;
  int vec;
  int CBPY;
  int CBPCM;
  int MODB;
  int CBPB;
  int COD;
  int header;
  int DQUANT;
  int total;
  int no_inter;
  int no_inter4v;
  int no_intra;
/* NB: Remember to change AddBits(), ZeroBits() and AddBitsPicture()
   when entries are added here */
} Bits;

typedef void (*WriteByte) (int);

/* Global variable */
extern WriteByte WriteByteFunction;

/* Prototypes */
```

```
int CompressToH263(CParam *params, Bits *bits);
int InitCompress(CParam *params);
void CloseCompress(CParam *params);
void SkipH263Frames(int frames_to_skip);

/* Procedure to detect motion, expects param->EncodeThisBlock is set to
   array.
   Advised values for threshold: mb_threholds = 2; pixel_threshold = 2 */
int FindMotion(CParam *params, int mb_threshold, int pixel_threshold);

#endif
```