Gici Emporda manual

(version 1.0)

GICI group

Department of Information and Communications Engineering

Universitat Autònoma Barcelona

http://www.gici.uab.es - http://gici.uab.cat/GiciWebPage/downloads.php

October 2011

Contents

1 Overview

Emporda is an implementation of the future CCSDS Recommended Standard for multispectral and hyperspectral image coding [1]. A small explanation about how this standard works, and some results about the performance it can achieve can be seen in this article [2]. The CCSDS Recommended Standard we are referring to is still a draft or a red book (CCSDS 123.0-R-1) and it is expected that will become a CCSDS Recommended Standard this fall. The aim of Emporda is to provide an implementation that can be useful as a reference in order to study how the algorithms work, to make a new different implementation, or to check the performance the future Recommended Standard can achieve.

Emporda provides two different modes, the compression mode and the decompression mode. In the compression mode, Emporda expects an image and generates a compressed file compliant with the Recommendation, as Emporda follows exactly all the practices recommended in the documentation of the standard and it has not extra or additional features. In the decompression mode, Emporda expects a compressed file, and recovers the original image. It must be noticed that only files with the correct header can be decompressed.

There is an important amount of parameters that can be set for the input image, the compression/decompression algorithms and for the coding/decoding algorithms, so it is important to know which is their effect in the performance of the algorithm. Emporda has by default the values of the parameters recommended in the documentation, but it is able to read a configuration file overwriting these values. Because there are some combinations of values for the parameters that are thought to be useful for some kind of images, we provide some files that can be used to set those parameters, and different examples of how should be set some of them. Another important feature is that Emporda checks the values of every parameter it receives and the combinations between them, to make sure that no errors are introduced. If there is an error, Emporda shows the correct values for the parameters causing the error, so the error can be fixed easily.

The source code of Emporda is publicly available in the web page of the group http://www.gici.uab.cat/Emporda, where some manuals can be found too. In order to guarantee the free distribution, Emporda has the General Public License (GNU GPL v3) of the Free Software Foundation (http://www.fsf.org/licensing/licenses/gpl.html).

Emporda has been designed to be modular, so the source code is quite flexible, allowing the user to easily integrate some modules to other applications. The main modules are the compressor, and the entropy coder. The implementation has been done this way to be able to test the performance of the compressor with other kind of encoders, and to make other implementations of other algorithms using the entropy coder as the encoder.

For all of these, it can be noticed that Emporda is an implementation that can be useful as a reference to study how the Recommended Standard is supposed to work, and as a guide to make an own implementation or testing the performance of the algorithm, but it is has not been designed to be an efficient implementation, so for commercial purposes the GICI group does not recommend the use of this application. However, all suggestions and comments about Emporda are welcome in order to improve the implementation, that can be reported to us (gici-dev@abra.uab.es).

We hope you enjoy it,

GICI group

2 Usage

2.1 Emporda's Requirements

The following section explains Emporda's requirements to be executed correctly, how it is been organized and how to use it. However, next sections will made explicit the parameters usage and will introduce some useful examples.

The basic requirements to execute Emporda can be divided into hardware requirements and software requirements. Hardware requirements are basically in terms of the system memory where it will be executed. Emporda needs an amount of memory that depends on the geometry of image, in which order the image samples are handled and the number of bands used in prediction. Anyway, Emporda does not load a whole image in memory, so an amount of ram close to the size of the image that is going to be compressed or decompressed is far enough. This observation is basically done because we consider that this program will work with images of big dimensions, and this can be problematic in some cases.

The input image is meant to be a three-dimensional array of signed or unsigned integer sample values. On software requirements the development of it has been under a set of libraries from Java Standard Edition version 1.6. On testing stage we have used J2SE version 1.6, so it means that it works over Java Runtime Environment 1.6 or higher. We suppose that it can run as well with version 1.5.

2.2 Emporda's Execution Overview

The program is organized in only one application called emporda.jar that is in dist directory. However, the source code is provided too, so anyone can compile its own jar using the command *ant*. The program's structure is modular which provides scalability and it lets the code to be used in other projects easily. There are two basic modules, the predictor and the coder which are used independently.

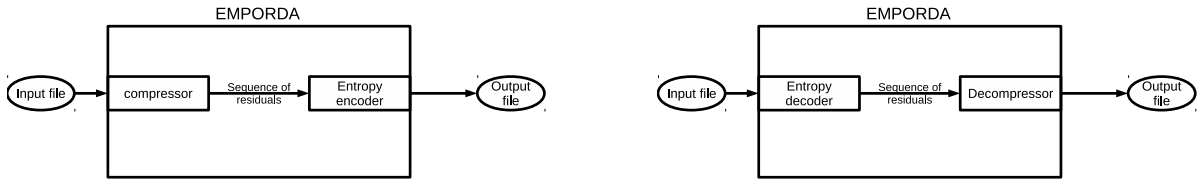

Table 1: Process of Emporda, in compression and decompression modes

To compress an image it is need the name of the image and its geometry (the number of bands, the number of rows and the number of columns). The rest of parameters will depend on the user preferences. This parameters can be for example,the name of the output file or the name of the options file (if needed). The following examples show how to use them (if the option file name is not used, the parameter "-f optionsFileName" must be removed):

```
  $java -jar emporda.jar -i inputImageName -ig numberOfBands numberOfRows
numberOfColumns 3 0 -o outputFileName -c -f optionFileName -v
```

When decompressing, Emporda only needs the name of the compressed file and the name of the image that the program will output. The following line specifies how to do it:

```
  $java -jar emporda.jar -i inputCompressedFile -o outputImageDecompressed
-d -v
```

In case a larger amount of memory is needed to be used by the Java Virtual Machine (increase the heap size) it must be add the parameter -Xmx{size} for increasing the maximum and/or -Xms{size} for increasing the minimum memory allowed. For example, if the user wants to compress an image as in the first example but wants to have at least 2GB and a maximum of 4GB of available memory:

```
   $java -Xms2048m -Xmx4096m -jar emporda.jar -i inputImageName -ig
numberOfBands numberOfRows numberOfColumns 3 0 -o outputFileName -c -f
optionFileName -v
```

3 Parameters

3.1 Application Parameters

The following section explains each parameter for the command line execution. It must be noticed that there are two different modes, compression and decompression modes, and the parameters that can be set depend on the mode that is going to be used. In the next frame the format of a table to show the information about the parameters is shown.

```
   Parameter:  how the option can be set.  There are two options:
-shortParameter or --longParameter
   Type:  type of the argument of the parameter
   Mandatory:  If the parameter is mandatory or it is not
   Description:  A small brief of the parameter
   Default value:  Default value set in the implementation
```

Parameter	**-h --help**
Type	
Mandatory	No
Description	Displays help and exits program.
Default Value	

Parameter	**-c --compress**
Type	
Mandatory	Yes for the compression mode
Description	Compression mode for the input image
Default Value	

Parameter	**-d --decompress**
Type	
Mandatory	Yes for decompression mode
Description	Decompression mode for the input file
Default Value	

Parameter	**-i --inputImage**
Type	string
Mandatory	Yes
Description	The input file: COMPRESSION: a 3D raw image DECOMPRESSION: a 3D image compressed with the Standard
Default Value	

Parameter	`-o --outputFile`
Type	string
Mandatory	Yes
Description	Output file COMPRESSION: Output file name without extension DECOMPRESSION: Output image file WITH extension
Default Value	

Parameter	`-ig --inputImageGeometry`
Type	int int int int int
Mandatory	Yes for compression mode.
Description	Geometry of the input image. Parameters are: 1- zSize (number of image components) 2- ySize (image height) 3- xSize (image width) 4- data type. Possible values are: 1- unsigned int (1 byte) 2- unsigned int (2 bytes) 3- signed int (2 bytes) 5- 1 if 3 first components are RGB, 0 otherwise
Default Value	

Parameter	`-so --sample-order`
Type	int
Mandatory	No
Description	In compression mode it sets in which order the pixels of the image will be read. In decompression mode sets in which order the pixels of the image recovered will be saved. Its options are: 0.- BSQ 1.- BIL 2.- BIP
Default Value	0

Parameter	`-v --verbose`
Type	
Mandatory	No
Description	Displays information about the progress in the task done by the program.
Default Value	

Parameter	`-e --endianess`
Type	int
Mandatory	No
Description	In compression mode it sets which endianess is considered for the bytes of every pixel of the image that will be read. In decompression mode sets in which order the bytes of the pixels of the image recovered will be saved. Its options are: 0.- big endian 1.- little endian
Default Value	0

Parameter	`-f --option-file`
Type	String
Mandatory	No
Description	Sets the option file name for the compression process.
Default Value	

Parameter	`-dbg --debug-mode`
Type	
Mandatory	No
Description	Outputs useful information for debugging the application
Default Value	

Parameter	`-format --pixel-format`
Type	int
Mandatory	No
Description	Sets the format in which the pixels of the decompressed image will be stored Values: 1.- byte 2.- unsigned integer (2 bytes) 3.- signed integer (2 bytes)
Default Value	

3.2 Option File Parameters

The following sections defines and specifies how to use parameters that can be set in the Recommended Standard. These options must be set in an option file. An example option file is distributed with the source code as a guide. The next frame shows how the information of every parameter will be displayed in the next sections.

```
   Parameter:  ParameterValue
   Range:  Range of values allowed for the parameter
   Description:  A small brief of the parameter
   Number of bits:  Number of bits needed to store the parameter in the
header of the file
   Default value:  Default value set in the implementation
```

3.2.1 Image Options

These options are relative to the properties of the image that is going to be compressed, so they may be adapted for every image that must be compressed.

Parameter	`DYNAMIC_RANGE`
Range	[2, 16]
Description	Maximum bit size for sample
Number of bits	4 bits
Default Value	16

Parameter	`SAMPLE_ENCODING_ORDER`
Range	[0, 1]
Description	Encoding Order: 1 = BSQ, 0 = BI
Number of bits	1 bit
Default Value	1

Parameter	SUBFRAME_INTERLEAVING_DEPTH
Range	[1, numBands]
Description	Subframe interleaving depth used in BI encoding.
Number of bits	16 bits
Default Value	0

Parameter	OUTPUT_WORD_SIZE
Range	[1,8]
Description	Integer number of output words.
Number of bits	3 bits
Default Value	4

Parameter	ENTROPY_CODER_TYPE
Range	[0, 1]
Description	Entropy coder type used: 0 Sample Adaptive, 1 Block Adaptive
Number of bits	1 bit
Default Value	0

3.2.2 Algorithm Options

Parameters that are used by the compressor for the prediction of a pixel.

Parameter	NUMBER_PREDICTION_BANDS
Range	[0, 15]
Description	Number of prediction bands.
Number of bits	4 bits
Default Value	15

Parameter	PREDICTION_MODE
Range	[0, 1]
Description	Prediction mode that will be used by the compressor. Full prediction mode uses information of the same band as the pixel that is going to be predicted, not as in reduced prediction mode 0 Full prediction mode, 1 Reduced prediction mode
Number of bits	1 bit
Default Value	0

Parameter	LOCAL_SUM_MODE
Range	[0, 1]
Description	The local sum mode tells the compressor the size of the neighborhood that will be used to predict a pixel 0 Neighbor oriented sum mode, 1 Column oriented sum mode
Number of bits	1 bit
Default Value	0

Parameter	REGISTER_SIZE
Range	[32, 64]
Description	Size of the register 6 bits.
Number of bits	6 bits
Default Value	32

3.2.3 Weight Options

Options used by the compressor relative to the weight vector. This vector is used to maintain causal information about the pixels of the image that have already been handled.

Parameter	WEIGHT_COMPONENT_RESOLUTION
Range	[4,19]
Description	Resolution of weight components.
Number of bits	4 bits
Default Value	13

Parameter	WEIGHT_UPDATE_SECI
Range	[4,11]
Description	Weight update scaling exponent change interval. Value is expressed as $\log_2$.
Number of bits	4 bits
Default Value	6

Parameter	WEIGHT_UPDATE_SE
Range	[-6, WEIGHT_UPDATE_SEFP]
Description	Weight update scaling exponent initial parameter .
Number of bits	4 bits
Default Value	-1

Parameter	WEIGHT_UPDATE_SEFP
Range	[WEIGHT_UPDATE_SE, 9]
Description	Weight update scaling exponent final parameter.
Number of bits	4 bits
Default Value	3

Parameter	WEIGHT_INITIALIZATION_METHOD
Range	[0, 1]
Description	Weight initialization method: 0 Default Initialization, 1 Custom Initialization
Number of bits	1 bit
Default Value	0

Parameter	WEIGHT_INITIALIZATION_TF
Range	[0, 1]
Description	Weight initialization Flag: 0 Not included in metadata, 1 Otherwise
Number of bits	1 bit
Default Value	0

Parameter	WEIGHT_INITIALIZATION_RESOLUTION
Range	if WEIGHT_INITIALIZATION_TABLE_FLAG is 0 $\rightarrow$ [0] Otherwise $\rightarrow$ [3, WEIGHT_COMPONENT_RESOLUTION+3]
Description	Weight initialization resolution.
Number of bits	5 bits
Default Value	0

Parameter	WEIGHT_INITIALIZATION_TABLE
Range	If this table is used, and $Q = WEIGHT\_INITIALIZATION\_RESOLUTION$ the values of this table must be in the range $[-2^Q, 2^Q - 1]$
Description	This table has the values used in the custom weight vector initialization.
Number of bits	every element Q bits
Default Value	0

3.2.4 Entropy Coder Options

Parameters relative to the entropy coder.

Parameter	UNARY_LENGTH_LIMIT
Range	[8, 32]
Description	Unary length limit.
Number of bits	5 bits
Default Value	16

Parameter	RESCALING_COUNTER_SIZE
Range	[4, 9]
Description	Rescaling counter size.
Number of bits	3 bits
Default Value	6

Parameter	INITIAL_COUNT_EXPONENT
Range	[1, 8]
Description	Initial count exponent.
Number of bits	3 bits
Default Value	1

Parameter	ACCUMULATOR_INITIALIZATION_TF
Range	[0, 1]
Description	Accumulator initialization table: 0 Not included in metadata, 1 Otherwise
Number of bits	1 bit
Default Value	0

Parameter	ACCUMULATOR_INITIALIZATION_CONSTANT
Range	[0, DYNAMIC_RANGE-2]
Description	Accumulator initialization constant.
Number of bits	4 bits
Default Value	5

Parameter	ACCUMULATOR_INITIALIZATION_TABLE
Range	If this table is used, and $D = DYNAMIC\_RANGE$ the values of this table must be in the range $[0, 2^D - 2]$
Description	This table has the values used in the sample coding algorithm.
Number of bits	every element D bits
Default Value	0

3.2.5 Block entropy Coder Options

Parameters relative to the block entropy coder.

Parameter	BLOCK_SIZE
Range	Must be 8 or 16
Number of bits	2 bits
Description	Block Size.
Default Value	16

Parameter	REFERENCE_SAMPLE_INTERVAL
Range	[1, 256]
Description	Reference sample interval.
Number of bits	12 bits
Default Value	1

4 Examples

4.1 Execution of Emporda

In this section we show some examples about how Emporda should be executed, and combinations of the different options in the command line.

- See the help information

```
$ java -jar dist/Emporda.jar --help
```

- Compress the image in input-file and save the result in output-file using the values by default in the Emporda implementation

```
$ java -Xmx1200m -jar dist/emporda.jar -i input-file \
-ig bands rows columns 3 0 -o output-file -c
```

- Compress the image in input-file and save the result in output-file, using a configuration file to overwrite some values set by default.

```
$ java -Xmx1200m -jar dist/emporda.jar -i input-file \
-ig bands row columns 3 0 -o output-file \
-c -f option-file
```

- Compress the image in input-file and save the result in output-file with a information of the progress of the process, using an option file.

```
$ java -Xmx1200m -jar dist/emporda.jar -i input-file \
-ig bands row columns 3 0 -o output-file \
-c -f option-file -v
```

- Decompress the image in input-file and save the result in output-file.

```
$ java -Xmx1200m -jar dist/emporda.jar -i input-file -o output-file -d
```

4.2 Parameters values

In this section we show some configuration of parameters than can be used to compress some kind of images. These options can be given to Emporda saving them in to a option file (option.txt) and using the option -f (-f option.txt). However, if more accurate explanation of the parameters is necessary, the CCSDS Red Book [1] should be seen, due to all the parameters and their effects in the compression and encoding processes are perfectly detailed.

- It is supposed that for hyperspectral uncalibrated images, the best results are achieved using the reduced prediction mode and the column oriented sum mode:

```
PREDICTION_MODE = 1
LOCAL_SUM_MODE = 1
```

- It is supposed that for hyperspectral calibrated images, the best results are achieved using the full prediction mode and the neighbor oriented sum mode:

```
PREDICTION_MODE = 0
LOCAL_SUM_MODE = 0
```

- It must be noticed when using the weight initialization table, that the bigger number that can be used in that table is indicated by the weight initialization resolution. So if it is 6, then the range of the values is in the range $[-32, 31]$.
- if the weight initialization table for a neighbor oriented mode with full prediction mode is used, then the table must have the same number of lines as the number of bands of the image. And in every line, there must be as many values as bands that can be used for the compression of the band plus three. For example:

```
PREDICTOR_METADATA_FLAG=1
WEIGHT_INITIALIZATION_RESOLUTION=4
WEIGHT_INITIALIZATION_METHOD=1
WEIGHT_INITIALIZATION_TF=1
NUMBER_PREDICTION_BANDS = 3
PREDICTION_MODE = 0
LOCAL_SUM_MODE = 0
WEIGHT_INITIALIZATION_TABLE = [0, 1, 2] \
[0, 1, 2, 3] \
[0, 1, 2, 3, 4] \
[0, 1, 2, 3, 4, 5] \
[0, 1, 2, 3, 4, 5] \
.
.
.
[0, 1, 2, 3, 4, 5]
```

- if the weight initialization table for a reduced prediction mode is used, then the table must have the same number of lines as the number of bands of the image. And in every line, there must be as many values as bands that can be used for the compression of the band. For example:

```
PREDICTOR_METADATA_FLAG=1
WEIGHT_INITIALIZATION_RESOLUTION=4
WEIGHT_INITIALIZATION_METHOD=1
WEIGHT_INITIALIZATION_TF=1
NUMBER_PREDICTION_BANDS = 3
PREDICTION_MODE = 1
WEIGHT_INITIALIZATION_TABLE = [] \
[0] \
[0, 1] \
[0, 1, 2] \
[0, 1, 2] \
.
.
.
[0, 1, 2]
```

- if the accumulator initialization table is used, then the accumulator initialization constant must be 0. It must be noticed too, that is difficult to know which values to use in the accumulator initialization table to achieve good compression results. The accumulator initialization table is a list with a value for every band of the image

```
ENTROPYCODER_METADATA_FLAG = 1
ACCUMULATOR_INITIALIZATION_TF = 1
ACCUMULATOR_INITIALIZATION_CONSTANT = 0
ACCUMULATOR_INITIALIZATION_TABLE = 0 1 2 3 4 5 6 7 8 \
9 10 11 12 13 14 12 0 \
.
.
.
9 10 11 12 13 14 12 0 \
```

- Observation: all values for the parameters used in the compression process are stored in the header of the generated file, except for the WEIGHT_INITIALIZATION_TABLE and the ACCUMULATOR_INITIALIZATION_TABLE which are optional. This means that if any of these tables are not stored in the header of the generated file, the option file used in compression is needed in the decompression process. If an option file is used in decompression, only the tables mentioned above can be read, due to all other parameters are already stored on the header of the generated file. As an example, if we set:

```
ENTROPYCODER_METADATA_FLAG = 1
ACCUMULATOR_INITIALIZATION_TF = 0
ACCUMULATOR_INITIALIZATION_CONSTANT = 0
ACCUMULATOR_INITIALIZATION_TABLE = 0 1 2 3 4 5 6 7 8 \
9 10 11 12 13 14 12 0 \
.
.
.
9 10 11 12 13 14 12 0 \
```

Then the ACCUMULATOR_INITIALIZATION_TABLE is not stored in the header of the generated file. So the same option file must be used in decompression. Compression would be:

```
$ java -Xmx1200m -jar dist/emporda.jar -i input-file \
-ig bands row columns 3 0 -o compressed-file \
-c -f option-file -v
```

And decompression could be achieved with:

```
$ java -Xmx1200m -jar dist/emporda.jar -i compressed-file \
-o output-file -d -f option-file -v
```

5 Notes

If you need further assistance, you might want to contact us directly.

References

[1] Consultative Committee for Space Data Systems (CCSDS), *Lossless Multispectral & Hyperspectral Image Compression CCSDS 123.0-R-1*, ser. Red Book (draft). CCSDS, May 2011. [Online]. Available: http://public.ccsds.org/sites/cwe/rids/Lists/CCSDS%201230R1/Attachments/123x0r1.pdf

[2] Jose Enrique Sánchez, Estanislau Augé, Josep Santaló, Ian Blanes, Joan Serra-Sagristà, Aaron Kiely, “Review and implementation of the emerging ccsds recommended standard for multispectral and hyperspectral lossless image coding,” *CCP 2011, Proceedings*, vol. 1, june 2011.