Implementation of MPEG-7 Descriptors:

Scalable Color Graphics

and

Group of Frames/Group of Pictures

Summary and Details

by

Thomas Bucsics

e0025908@stud3.tuwien.ac.at

1. Introduction

1.1 Motivation and details

This descriptor was implemeted by *Thomas Bucsics*[1] as a bachelor's project at the *Interactive Media Systems Group*[2] of the *Vienna University of Technology*[3].

It was developed in *Java*[4] using *Eclipse*[5] and is to be used with the *VizIR project*[6] content-based retrieval system based on the *Java Media Framework*[7].

The *ISO/IEC 15938-3 FCD* and *MPEG-7 Visual part of XM*[8]*, Version 8.0* and the *IEEE Transactions on Circuits and Systems for Video Technology, Vol. 11, No 6, June 2001* documents were used for reference and as specifications.

For building the class descriptions and automated documentation, *Javadoc*[9] was used.

The class diagram was generated using *Omondo EclipseUML Free Edition 2.1.1*[10].

JUnit[11] classes were used as superclasses for the test automation implementation.

1.2 General purpose of the Scalable Color Descriptor concept

The goal is to acquire a representation of the color composition of an image that is scalable in both the number of coefficients it contains and in the number of bits each coefficient is assigned. This is accomplished by doing the following:

The Scalable Color Descriptor extracts a quantized HSV color histogram from a given picture (usually in the RGB format). The probability values of each bin are calculated and indexed. The resulting histogram is then transformed via a discrete Haar transformation (as described in the *MPEG-7 Visual part of XM*[12]*, Version 8.0* document), nonuniformly quantized and offset, and the resulting array of values is then sorted.

The resulting values can now be assigned a lower bit resolution, and the number of coefficients can be reduced by half iteratively.

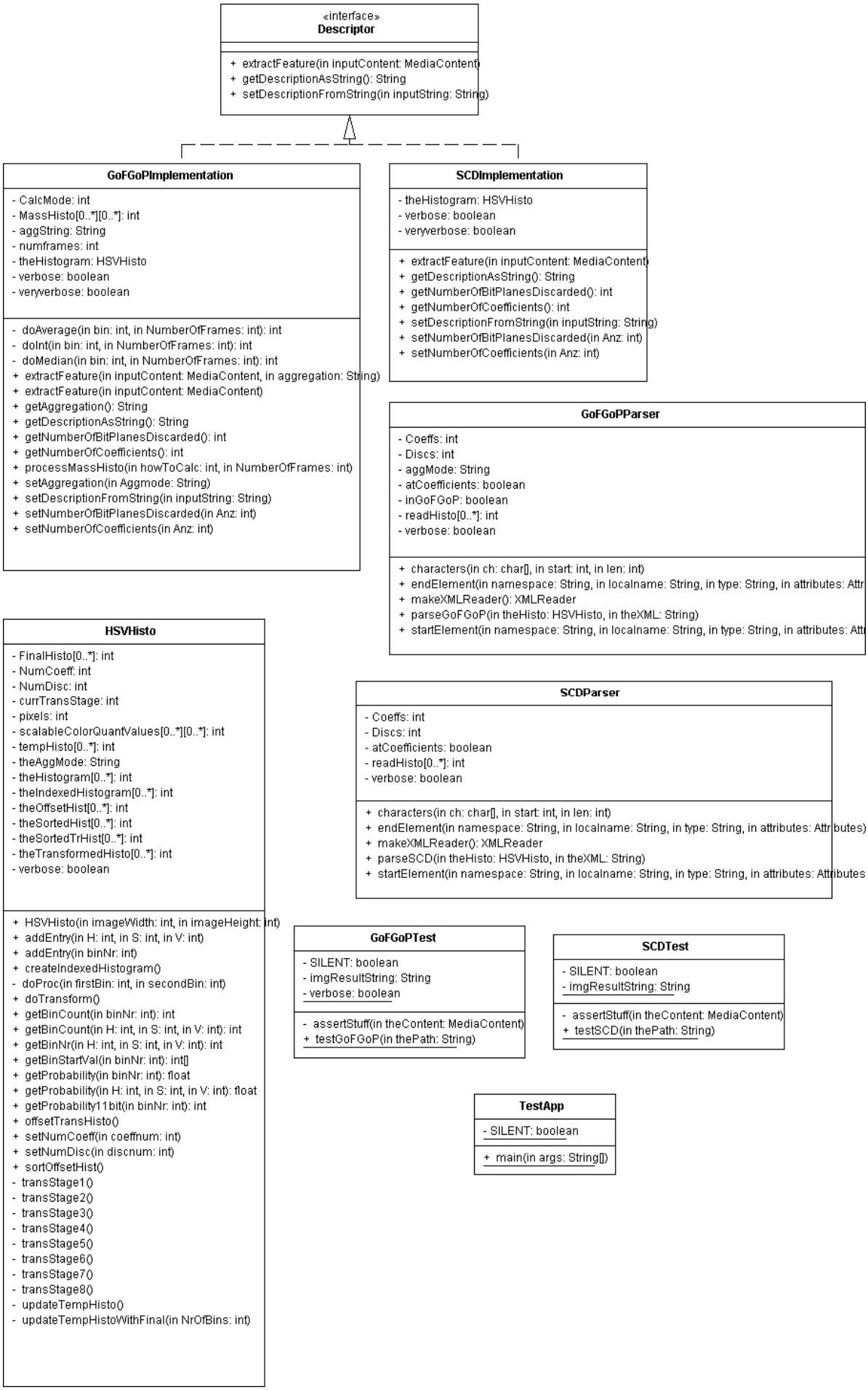
If information from more than one image is to be acquired, the same steps are taken, except that before indexing of the histogram, one histogram is acquired for each frame or picture. These are then merged into a single histogram by calculating either the average, median or intersection value for each bin.

The *SCDImplementation* and *GoFGoPImplementation* classes allow 16, 32, 64, 128 or all 256 values to be retained, and provides, in addition to the quantization described by the normative parts of the standard, the ability to discard from 0 (none) to 8 (all, retaining only the sign of the value) bits.

Descriptions can be read from a media object of the class *MediaContent,* be acquired form existing XML descriptions and be output to XML strings.

1e0025908@stud3.tuwien.ac.at
2http://www.ims.tuwien.ac.at
3 http://www.tuwien.ac.at
4http://java.sun.com
5http://www.eclipse.org
6http://cbvr.ims.tuwien.ac.at
7Http://java.sun.com/jmf
8http://www.lis.ei.tum.de/research/bv/topics/mmdb/e_mpeg7.html
9http://java.sun.com/j2se/javadoc
10http://www.omondo.com
11http://www.junit.org
12http://www.lis.ei.tum.de/research/bv/topics/mmdb/e_mpeg7.html

2. Class Diagram



3. The Scalable Color Graphics Descriptor

This descriptor is implemented in the *SCDImplementation* class contained in the file "SCDImplementation.java", and implements the *Descriptor* interface.

3.1 The *extractFeature* method

3.1.1 Creating the initial HSV histogram from an image

The *extractFeature* method of the *SCDImplementation* class takes a *MediaContent* object containing an image as a parameter and extracts its height and width. Then the method goes through all the pixels, projecting each one to HSV space by instantiating an object of the *FlexPixel* class which was designed specifically for that purpose, and adds a corresponding entry into a member object of the *HSVHisto* class, using its *addEntry* method, which first uniformly quantizes the H component into 16 bins, and the S and V components of the pixel into 4 bins each, thus calculating a corresponding 8 bit bin number. The counter for that bin is incremented by one each time *addEntry* is called with a component triplet corresponding to that bin or the with bin number itself.

3.1.2 Extracting an indexed probability histogram

After all the pixels are processed, the *createIndexedHistogram* method of that *HSVHisto* object is called, which gets the probability for each bin, encoded in an eleven-bit value (using the *getProbability11bit* method), and then nonuniformly quantizes these probability values into four-bit values according to the table provided in the ISO specification[13].

3.1.3 Tansforming the indexed histogram

Now the *doTransform* method of the *HSVHisto* object is called, which goes through the eight transformation stages depicted in the ISO specification[14], each in its own method (*transStage1* through *transStage8*).

3.1.4 Offsetting and quantizing the transformed histogram

The next step calls the *offsetTransHisto* method, which uses a table from the specifications[15] to offset and nonuniformly quantize the transformed histogram bins. For each bin, a specific offset and bit allocation number is used.

3.1.5 Sorting the histogram

The last step of *extractFeature* call the *sortOffsetHist* method of the *HSVHisto* class. This goes through all bins and creates a new array with the values from the offset and quantized histogram, but with the order specified in the ISO specification[16].

3.2 The get/set methods

3.2.1 setNumberOfCoefficients

This method calls the *setNumCoeffs* method of the *HSVHisto* object, which drops the last *n* entries in the histogram according to the semantics specified in the ISO specification[17].

13ISO/IEC 15938-3 FCD page 42, Table 11
14MPEG-7 Visual Part of XM, Version 8.0, page 25, Figure 16
15ISO/IEC 15938-3 FCD page 41, Table 10
16ISO/IEC 15938-3 FCD page 40
17ISO/IEC 15938-3 FCD page 39

3.2.2 setBitplanesDiscarded

This method calls the *setNumDisc* method of the *HSVHisto* object, which either moves each value in the histogram lower towards the LSB (if the new bitnumber is lower than the old one), or moves the values up towards the MSB (if the new bitnumber is higher than the old one), according to the semantics specified in the ISO specification[18].

3.2.3 getNumberOfCoefficients

Returns the *numCoeff* member variable of the *HSVHisto*.

3.2.4 getBitplanesDiscarded

Returns the *numDisc* member variable of the *HSVHisto*.

3.3 The XML interfaces

3.3.1 setDescriptionFromString

This method instantiates an *SCDParser* object, which in turn uses a *SAXParser* to read the relevant data for filling the information needed to have a workable *HSVHisto* object for the *SCDImplementation*. The *SCDParser* class is also a subclass of the *DefaultHandler* class and implements the *ContentHandler* interface. A *StringReader* object is created on the input string, which then is used to create an InputSource object which is then fed into the *SAXParser*, using the *SCDParser* as the *ContentHandler*. Simple boolean flags are used to track the current position inside of the XML description that is parsed.

3.3.2 getDescriptionAsString

This method creates a string by stringing together the XML description standard specified in the ISO specification[19] and the relevant data read from the *HSVHisto* object.

4. The Group of Frames/Group of Pictures Descriptor

This descriptor is implemented in the *GoFGoPImplementation* class contained in the file “GoFGoPImplementation.java”, and implements the *Descriptor* interface.

4.1 The *extractFeature* method

This method works like the *extractFeature* method of the *SCDImplementation* descriptor, but takes a *MediaContent* object containing more than one frame as an input.

It then goes through each frame, converting, as in *SCDImplementation*, to HSV, thus creating a histogram for each frame. The histograms of all frames are then all merged, using the *processMassHisto* method, which, by default, uses the median of all the frames for one specific bin as the value that is entered into the histogram that will then be used for further processing.

The resulting histogram is then processed like it would have been in the *SCDImplementation* class, meaning that it is indexed, transformed, offset and sorted.

If, in addition to a *MediaContent* object, an aggregation string (“Average”, “Median” or “Intersection”) is passed as a parameter, the description will be extracted using that aggregation type instead of the default “Median”.

18ISO/IEC 15938-3 FCD page 39
19ISO/IEC 15938-3 FCD page 38

4.2 The get/set methods

All get/set methods from *SCDImplementation* also exist for the *GoFGoPImplementation*, but there are two additional ones:

4.2.1 getAggregation

This method returns the *aggString* member variable from the *GoFGoPImplementation* object, referring to one of the three aggregation methods described in the specification[20]:

“Average”, “Median”, “Intersection”, referring to the method that is used to merge the histograms from all the frames in the *MediaContent* object that is processed into one histogram that is then treated like it would be in the *SCDImplementation*.

4.2.2 setAggregation

This method takes a string as parameter, which should be one of three aggregation methods described in the specification[21]:

“Average”, “Median”, “Intersection”, referring to the method that is used to merge the histograms from all the frames in the *MediaContent* object that is processed into one histogram that is then treated like it would be in the *SCDImplementation*.

The histograms created when *extractFeature* was first called are again merged and processed when this method is called.

4.3 The XML interfaces

The XML methods (*getDescriptionAsString* and *setDescriptionFromString*) work the same way the corresponding methods do in the *SCDImplementation*, meaning that the output string is created by stringing together XML tags and relevant variables, and that the parsing is done by feeding the input string to a *SAXParser*, after creating a *StringReader* and an *InputSource* from it.

5. The test application and JUnit tests

The test application *TestApp* contained in the file “TestApp.java” takes the location of the “Media” folder that should contain test.jpg and test.avi provided by this test suite as an argument, and then runs *SCDTest.testSCD* and *GoFGoPTest.testGoFGoP* to check the most important features of the classes.

Both *SCDTest* and *GoFGoPTest* are subclasses of the *JUnit* class *TestCase*.

5.1 SCDTest.testSCD

This method creates a *MediaContent* object on the file “test.jpg” in the “Media” folder that should be provided by this test suite, then uses the *SCDImplementation* to extract an SCD Description, which is then read into a string using the *getDescriptionAsString* method and compared by *assertEquals* to a hardcoded member string of the *SCDTest* class.

Then the *setDescriptionFromString* method is tested by using it to read the hardcoded string into the *SCDImplementation*, which is again extracted using *getDescriptionAsString* and again compared via *assertEquals* to the original string.

20MPEG-7 Visual Part of XM, Version 8.0, page 32
21MPEG-7 Visual Part of XM, Version 8.0, page 32

5.2 GoFGoPTest.testGoFGoP

This method creates a *MediaContent* object on the file “test.avi” in the “Media” folder that should be provided by this testsuite, then uses the *GoFGoPImplementation* to extract a GoFGoP Description, which is then read into a string using the *getDescriptionAsString* method and assured that it is not null. Comparison via *assertEquals* to another string is currently not possible (see *“5. Known shortcomings and bugs”*).

Then the *setDescriptionFromString* method is tested by using it to read the hardcoded string into the *GoFGoPImplementation*, then again extracted using *getDescriptionAsString* and again compared via *assertEquals* to the original string.

6. Known shortcomings and bugs

6.1 Internal shortcomings and bugs

6.1.1 Attributes are reset to default by extractFeature

Currently the set attributes that were set using *setNumberOfCoefficients*, *setBitplanesDiscarded* or *setAggregation* are reset to their default values each time extractFeature is called, to avoid complications with previously extracted descriptions.

The aggregation of *GoFGoPImplementation.extractFeature* can be specified (see 3.1), since changing it later would require processing every frame's histogram anew.

6.1.2 Changing attributes with a histogram parsed from an XML string

SetNumberOfCoefficients and *setBitplanesDiscarded* should work the same way as with a description obtained from a *MediaContent* using *extractFeature*, even with histograms that are parsed from an XML string.

setAggregation however, cannot work correctly because only the merged, transformed, offset and sorted histogram is contained in the XML string, and all the raw histograms from all the frames would be necessary to reaggregate.

Also, *doTransform*, *offsetTransHisto* and *sortOffsetHist* cannot work, because the raw histograms are not available when using parsed information.

Please note that if you *do* call these methods after having first read a description via *extractFeature* and then *setDescriptionFromString*, the description from the previous *extractFeature* might be reextracted and the one that was read by *setDescriptionFromString* might be overwritten.

6.2 Shortcomings and bugs due to use of external classes

6.2.1 Non-deterministic behaviour of GoFGoPImplementation.extractFeature

When extracting a description twice from the same MediaContent using the same video/group of pictures, the results may vary slightly. The author has been unable to trace the exact source of these discrepancies, but is currently led to believe that this is due to the way the *MediaContent* class or the *Java Media Framework*[22] class reads frames, which seems to have something to do with CPU usage at the moment the frames are read.

This is also the reason why the string produced by *getDescriptionAsString* of the *GoFGoPImplementation* object in the *GoFGoPTest.testGoFGoP* is currently not compared

22Http://java.sun.com/jmf

by *assertEquals* to another string, but is only tested by *assertNotNull*.

If anybody can give any details on this issue, please contact the author of this document.

6.3 Inconsistencies with XM[23]

The results this suite provides might differ from results extracted using the XM library. Since XM doesn't seem to be normative, the author currently cannot tell what the cause of this is. Results from this suite seem plausible, and so do XM's results. It might be that some of the specifications were misinterpreted, or that unknown bugs are the cause of this.

If anybody can give any details on this issue, please contact the author of this document.

23http://www.lis.ei.tum.de/research/bv/topics/mmdb/e_mpeg7.html