CREATIVE MATH.
14 (2005), 00 - 00

A Java implementation of the SPIHT coder

Ovidiu Cosma and Vasile Lupşe

Abstract. This article presents a Java implementation of one of the best coding algorithms for the subbands of images. The SPIHT (Set Partitioning in Hierarchical Trees) algorithm performs a progressive coding of the transformed images, speculating the correlation between the DWT coefficients, with a data structure called zerotree.

1. Introduction

Some of the best coding algorithms for image subbands speculate the correlations between the DWT coefficients with a data structure called zerotree. A zerotree is a quad-tree with all the nodes smaller or equal with the root. A zerotree that contains only insignificant coefficients, can be coded with a single symbol, and will be completed with zeros by the decoder. Thus the correlations between the DWT coefficients in successive subbands can be efficiently speculated. EZW (Embedded image coding using Zerotrees of Wavelet coefficients) [1],[4] is the first algorithm based on zerotrees. EZW performs a progressive coding of the transformed image into an embedded code, which has the property that all the encodings of the same image at lower bit rates are embedded in the beginning of the bit stream for the target bit rate. The coding process can be stopped at any bit rate, or it can be continued until the coefficients of the DWT are represented with a precision however high.

2. The SPIHT algorithm

The SPIHT algorithm codes an image in several passes. Each of them has an associated threshold p. For the first pass, p is initialized with the largest power of two, which is smaller or equal with the largest DWT coefficient. At each of the passes, the coefficients whose absolute values reach or exceed the threshold become significant. They are moved in a List of Significant Coefficients (LSC), and then the threshold p is halved. The process is ended when the bit rate reaches a target value.

The SPIHT coding algorithm is based on the following data structures:

LIS (List of Insignificant Sets) that contains the roots of all the zerotrees, *LIP* (List of Insignificant Pixels) that contains insignificant coefficients that do not belong to any of the zerotrees and *LSP* (List of Significant Pixels) that contains coefficients that became significant at the current pass, or at one of the previous passes.

Received: 18.05.2005. In revised form: 20.06.2005.
2000 *Mathematics Subject Classification.* 94A08.
Key words and phrases. *Image compression, coding of image subbands.*

The LIS and the LIP are initialized with all the coefficients of the first subband. At each pass, the LIP and the trees in the LIS are scanned for significant coefficients. If a significant coefficient is found in the LIP, it is moved to the LSP. The trees in the LIS that contain one or more significant coefficients are divided into subtrees with all the coefficients smaller than the current threshold p. The coefficients that remain outside the new zerotrees are placed in the LIP or in the LSP if they are smaller respectively larger than p.

The SPIHT algorithm is described in detail in [1],[2],[3].

3. The coding application

The image subband coding is performed by the *codeImage* method of the *SPIHT* class. The image data is taken from the unidimensional array *tab*, and the resulting code is written to the output stream *dos*. The *List* class implements a simple linked list to hold the LIS. The elements of the list are of type *Node*.

```
package compresie;
import java.io.*;
class SPIHT{
//———-
int dim;//side of the image
float bpp;//keeps the target bitrate in bits per pixel
int lungCod;//code length = bpp*dim*dim
float[] tab;//transformed image (the DWT coefficients)
float[] LIP;//List of Insignificant Pixels
float[] LSP;//List of Significant Pixels
List LIS;//List of insignificant sets
int nrLIP, nrLSP;//number of elements in the LIP and LSP
float n;//current threshold
DataOutputStream dos;//Stream the image code will be written to
SPIHT(float[] tab, float bpp, int dim, DataOutputStream dos){
//————————————————————
//constructor allocates memory and initializes the fields
this.dos = dos;
//length of the code at the target bitrate
lungCod = (int)(bpp*dim*dim);
this.tab = tab; this.dim = dim; this.bpp = bpp;
LIP = new float[dim*dim]; LSP = new float[dim*dim];
LIS = new List();
nrLIP = 0; nrLSP = 0;
}//end of constructor
boolean SD(int i, int j){
//———————
//verifies if the quadtrees with the roots in the direct
//descendents of coefficient(i,j) are zerotrees
return (i<dim/2 && j<dim/2) && (genTree(2*i, 2*j) ||
genTree(2*i, 2*j+1) || genTree(2*i+1, 2*j) || genTree(2*i+1, 2*j+1));
}//end of SD
```

```
boolean SL(int i, int j){
//————————————
//verifies if the quadtrees with the roots in the descendents of the
//descendents of coefficient(i,j) are zerotrees
return (i<dim/2 && j<dim/2) && (SD(2*i, 2*j) ||
SD(2*i, 2*j+1) || SD(2*i+1, 2*j) || SD(2*i+1, 2*j+1));
}//end of SL
boolean genTree(int i, int j){
//————————————————-
//verifies if the quadtree with the root in coefficient(i,j) is a //zerotree
if(Math.abs(tab[dim*i+j]) >= n)
return true;
else
return SD(i,j);
}//end of genTree
float retMax(){
//————-
//computes the initial threshold
float max = 0;
for(int i=0; i<dim*dim; i++)
if(Math.abs(tab[i])>max)
max = Math.abs(tab[i]);
float power = (float)Math.floor(Math.log(max)/Math.log(2));
return (float)Math.pow(2,power);
}//end of retMax
void treatCoef(int k, int l)throws EndCoding, IOException{
//————————————————————————
//code the coefficient at position (k,l)
float coef = tab[dim*k+l];
if(Math.abs(coef) >= n){
putBit(true);//code SN(k,l)=1
LSP[nrLSP++] = Math.abs(coef)-n;//add coefficient(k,l) to the LSP
putBit(coef < 0);//code the sign of the coefficient
}else{
putBit(false);//code SN(k,l)=0
LIP[nrLIP++] = coef;
}
}//end of treatCoef
void codeImage()throws IOException, EndCoding{
//————————————————————
noBitsCodif = 0;
//number of coefficients added to the LSP at the previous passes
int noLSPPrev = 0;
n = retMax();//initialize the threshold
//write the file header
dos.writeInt(dim);//side of the image
```

```
dos.writeFloat(tab[0]);//general average
dos.writeFloat(n);//initial threshold
dos.writeFloat(bpp);//encoding bitrate
//—step 1: Initialization
//Initialize the LIP
LIP[nrLIP++] = tab[1];
LIP[nrLIP++] = tab[dim];
LIP[nrLIP++] = tab[dim+1];
//Initialize the LIS
LIS.add(0,1,'A'); LIS.add(1,0,'A'); LIS.add(1,1,'A');
do{
//Step2: Sort
for(int i=0; i<nrLIP; i++)
//verifies if the element vas not eliminated from LIP
if(LIP[i] != Float.POSITIVE_INFINITY)
if(Math.abs(LIP[i]) >= n){
putBit(true);
//move the coefficient to the LSP
LSP[nrLSP++] = Math.abs(LIP[i])-n;
//The previous bit 1 indicates the most significant bit of the //coefficient. This coefficient will not be processed in the refinement //step. Next code the sign of the coefficient.
putBit(LIP[i] < 0);
//disables the coefficient in the LIP
LIP[i] = Float.POSITIVE_INFINITY;
}else
putBit(false);//the coefficient remains insignificant
boolean nextBit;//holds the next bit of the code
//process the LIS
for(LIS.start(); LIS.current != null; LIS.next()){
if(!LIS.current.valid)
continue;//step over the eliminated sets
if(LIS.current.type == 'A'){
nextBit = SD(LIS.current.i, LIS.current.j);
putBit(nextBit);
if(nextBit){
//process the descendents of (i,j)
treatCoef(2*LIS.current.i, 2*LIS.current.j);
treatCoef(2*LIS.current.i+1, 2*LIS.current.j);
treatCoef(2*LIS.current.i, 2*LIS.current.j+1);
treatCoef(2*LIS.current.i+1, 2*LIS.current.j+1);
if(4*LIS.current.i+3<dim && 4*LIS.current.j+3<dim){
//if the direct descendents of the current node have
//descendents
LIS.current.type = 'B';
LISTypeB();
```

```
}else
LIS.current.invalidate();
}//end if(nextBit)
}//end if(LIS.current.type == 'A')
else //the set is of type B
LISTypeB();
}//end the process of LIS
//step3: Refinement
for(int i=0; i<noLSPPrev; i++)
if(LSP[i] >= n){
putBit(true);
LSP[i] -= n;
}else
putBit(false);
noLSPPrev = nrLSP;//No of coefficients that will be processed at
//the next crossing of step 3
//step 4: Update the threshold
n /= 2;
}while(true);
}//end of codeImage
void LISTypeB()throws EndCoding, IOException{
//————————————————————-
//code a set of type B
boolean nextBit = SL(LIS.current.i, LIS.current.j);
putBit(nextBit);
if(nextBit){
LIS.add(2*LIS.current.i, 2*LIS.current.j, 'A');
LIS.add(2*LIS.current.i+1, 2*LIS.current.j, 'A');
LIS.add(2*LIS.current.i, 2*LIS.current.j+1, 'A');
LIS.add(2*LIS.current.i+1, 2*LIS.current.j+1, 'A');
LIS.current.invalidate();
}
}//end of LISTypeB
int noBitsCodif;//length of the code
byte byteCod;//holds maximum 8 bits of the code
void putBit(boolean theBit)throws EndCoding, IOException{
//————————————————————————-
//puts the next bit in the image code
byte mask = (byte)(1 « 7 - noBitsCodif % 8);
if(theBit)
byteCod |= mask;
else
byteCod &= ∼mask;
noBitsCodif++;
if(noBitsCodif % 8 == 0){
dos.writeByte(byteCod);
```

```
if (noBitsCodif >= lungCod){
dos.close();
throw new EndCoding("End of image coding");
}
}
}//end of putBit
}//end of class SPIHT
class EndCoding extends Exception{
//————————————————
public GataCitBiti(String s){ super(s); }
}//end class EndCoding
class List{
//————
Node first = null, last = null, current = first;
void add(int i, int j, char tip){//add a new node in the list
Node a = new Node(i,j,tip,last);
last = a;
if(first == null)
first = last;//add the first node in the list
}
void start(){ current = first; }//select the first node in the list
Node next(){//select the next node in the list
if(current != null)
current = current.next;
return current;
}
}//end of class List
class Node{
//————
int i,j;//hold the position of the root of the set
char type;//type of the set (A or B)
Node next;//link to the next node in the list
boolean valid;
Node(int i, int j, char type){
this.i = i; this.j = j; this.type = type;
next = null; valid = true;
}
Node(int i, int j, char type, Node last){
this(i,j,type);
if(last != null)
last.next = this;
}
void invalidate(){ valid = false; }
}//end of class Node
```

4. Conclusions

The following example codes the square image of side *DIM*, whose DWT coefficients are in array Y, at *bpp* bits per pixel, and places the resulting code in file *ImageCode*.

```
try{
DataOutputStream dos = new DataOutputStream(new BufferedOutputStream(
new FileOutputStream("ImageCode")));
new SPIHT(Y,bpp,DIM,dos).codeImage();
}
catch(IOException e){}
catch(EndCoding g){}
```

The following graph illustrates the processing speed of the implementation. A 1,4GHz PIV CPU was used for the tests. The test images were square, with the side of 512 pixels.

References

[1] Cosma O., *Contributions to the Coding of Image Subbands*, PhD Thesis, Polytechnic University, Bucharest 2003

[2] Cosma O., *The Implementation of a SPIHT Codec*, Buletinul Ştiinţific al Universităţii din Baia Mare, seria B, Matematică – Informatică, 2002

[3] Said A., Pearlman W. A., *A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, June 1996

[4] J. Shapiro M., *Embedded Image Coding Using Zerotrees of Wavelet Coefficients*, IEEE Transactions on Signal Processing, vol. 41 no. 12, 1993

North University of Baia Mare
Department of Mathematics and Computer Science
Victoriei 76, 430122 Baia Mare, Romania
E-mail address: cosma@alphanet.ro
E-mail address: vasilelupse@yahoo.co.uk