Теория и практика вейвлет-преобразования. ВОРОБЬЕВ В.И., ГРИБУНИН В.Г. ВУС, 1999. С.1-204.

Текст функций, выполняющих прямое и обратное вейвлет-преобразование изображения

```
// Функции для вычисления прямого и обратного вейвлет-преобразования
// изображения с использованием биортогональных 9/7 фильтров. Текст
// программы – модифицированная версия текста G.Davis
//(http://www.cs.dartmouth.edu/~Davis).
//  Выполнение вейвлет-преобразования изображения
// размером 512x512 на компьютере P-166 занимает примерно 0.4 с.


#include "wavelet.h"
#define al0 0.852699
#define al1 0.377403
#define al2 -0.110624
#define al3 -0.023849
#define al4 0.037829
#define ah0 0.788485
#define ah1 -0.418092
#define ah2 -0.040690
#define ah3 0.064539
#define sl0 -0.852699
#define sl1 -0.418092
#define sl2 0.110624
#define sl3 0.064539
#define sl4 -0.037829
#define sh0 -0.788485
#define sh1 0.377403
#define sh2 0.040690
#define sh3 -0.023849
#define AnaLowPass(v) al0*v[0]+al1*(v[1]+v[-1])+al2*(v[2]+v[-\
2])+al3*(v[3]+v[-3])+al4*(v[4]+v[-4])

#define AnaHighPass(v) ah0*v[0]+ah1*(v[1]+v[-1])+ah2*(v[2]+v[-\
2])+ah3*(v[3]+v[-3])
#define SynLowPass(v) sl0*v[0]+sl1*(v[1]+v[-1])+sl2*(v[2]+v[-\
2])+sl3*(v[3]+v[-3])+sl4*(v[4]+v[-4])
#define SynHighPass(v) sh0*v[0]+sh1*(v[1]+v[-1])+sh2*(v[2]+v[-\
2])+sh3*(v[3]+v[-3])
WaveletTransform::WaveletTransform ()
{
// Это необходимо для продолжения изображения при симметричном
отражении
npad = 4;
}
WaveletTransform::~WaveletTransform ()
{
}
```

```
void WaveletTransform::Reflection(float * h, float * t)
{
for (int i=1; i <= npad; i++)
{
h[-i] = h[i];
t[i] = t[-i];
}
}
// ---------------------------------------------------------------------------
// FWT (int ** input, float ** output, int hSize, int vSize,
// int trSteps, int symExtension)
//
// Функция прямого вейвлет-преобразования. Выполняется разделимое
преобразование, как описано в главе 9.
// Входные параметры:
// float *input: входной буфер данных,
// float *output: выходной буфер данных,
// int hSize: горизонтальный размер изображения,
// int vSize: вертикальный размер изображения,
// int trSteps: число шагов декомпозиции,
// int symExtension: вид продолжения на границе, по умолчанию -1.
// ---------------------------------------------------------------------------
BOOL WaveletTransform::FWT (float *coeff, int trSteps, int hSize, int vSize)
{

int i, j, lowSize, highSize;
int hTransSize = hSize;
int vTransSize = vSize;
float ** output = new float * [vTransSize];
output[0] = coeff;
for ( i=1; i<vSize; i++ )
output[i] = output[i-1] + hTransSize;
int *intIn, *intOut;
float *floatIn, *workingVector, *bufferPtr, *bufferEnd;
floatIn = new float [2*npad + Max(hTransSize, vTransSize)];
workingVector = floatIn+npad;
while (trSteps--)
{
// Выполняем свертку каждой строки с НЧ фильтром
for ( i=0; i<vTransSize; i++ )
{
lowSize = (hTransSize+1) >> 1;
highSize = hTransSize - lowSize;
// Копируем строку i в массив данных
memcpy( workingVector, output[i], hTransSize*sizeof(float) );
Reflection( workingVector, workingVector+hTransSize-1);
for ( bufferPtr = workingVector, j = 0; j < highSize; j++)
{
output[i][j] = AnaLowPass(bufferPtr);
bufferPtr++;
output[i][j+lowSize] = AnaHighPass(bufferPtr); bufferPtr++;
```

```
}
if ( lowSize > highSize )
output[i][highSize] = AnaLowPass(bufferPtr);
}
// Выполнение свертки НЧ фильтра со столбцами
for ( j=0; j<hTransSize; j++ )
{
lowSize = (vTransSize+1) >> 1;
highSize = vTransSize - lowSize;
// Копируем столбец j в массив данных
for ( i=0; i<vTransSize; i++ )
workingVector[i] = output [i][j];
Reflection( workingVector, workingVector+vTransSize-1);
for ( bufferPtr = workingVector, i = 0; i < highSize; i++)
{
output[i][j] = AnaLowPass(bufferPtr);
bufferPtr++;
output[i+lowSize][j] = AnaHighPass(bufferPtr); bufferPtr++;
}
if ( lowSize > highSize )
output[highSize][j] = AnaLowPass(bufferPtr);
}
hTransSize = (hTransSize + 1) / 2;
vTransSize = (vTransSize + 1) / 2;
}
delete output;
delete floatIn;
return TRUE;
}
// ----------------------------------------------------------------------------
// IWT (float *input, float *output, int hSize, int vSize,
// int trSteps, int symExtension)
//
// Обратное вейвлет-преобразование
// Входные параметры:
// float *input: входной буфер данных,
// float *output: выходной буфер данных,
// int hSize: размер изображения по горизонтали,
// int vSize: размер по вертикали,
// int maxOrig: максимальное значение пиксела в исх. изображении,
// int minOrig: минимальное значение пиксела в исх. изображении,
// int trSteps: число шагов декомпозиции,
// int symExtension: продолжение на границе, по умолчанию -1.
// ----------------------------------------------------------------------------
BOOL WaveletTransform::IWT( float *coeff, int trSteps, int hSize,
int vSize, int maxOrig, int minOrig)
{
int i, j;
float ** input = new float * [vSize];
input[0] = coeff;
for ( i=1; i<vSize; i++ )
```

```
input[i] = input[i-1] + hSize;
int * hTransSize = new int [trSteps];
int * vTransSize = new int [trSteps];
int hCrtSize, vCrtSize, normalSize, lowSize, highSize;
hTransSize[0] = hSize;
vTransSize[0] = vSize;
for (i = 1; i < trSteps; i++)
{
hTransSize[i] = (hTransSize[i-1] + 1) / 2;
vTransSize[i] = (vTransSize[i-1] + 1) / 2;
}
int *intIn, *intOut;
float *floatIn, *workingVector, *bufferPtr, *bufferEnd;
floatIn = new float [2*npad + Max(hSize, vSize)];
workingVector = floatIn+npad;
while (trSteps--)
{
hCrtSize = hTransSize[trSteps];
vCrtSize = vTransSize[trSteps];
// Выполняем реконструкцию для столбцов
for (j = 0; j < hCrtSize; j++)
{
lowSize = (vCrtSize + 1) >> 1;
highSize = vCrtSize - lowSize;
normalSize = highSize*2;
bufferPtr = workingVector;
for ( i = 0; i < highSize; i ++ )
{
*bufferPtr = input[i][j];
bufferPtr++;
*bufferPtr = input[i+lowSize][j];
bufferPtr++;
}
if ( lowSize > highSize )
workingVector[normalSize] = input[highSize][j];
Reflection(workingVector, workingVector + vCrtSize - 1);
bufferPtr= workingVector;
for ( i = 0; i < normalSize; )
{
input[i++][j] = SynHighPass(bufferPtr);
bufferPtr++;
input[i++][j] = SynLowPass(bufferPtr);
bufferPtr++;
}
if ( lowSize > highSize )
input[normalSize][j] = SynHighPass(bufferPtr);
}
// Выполняем реконструкцию для строк
for (i = 0; i < vCrtSize; i ++)
{
lowSize = (hCrtSize + 1) >> 1;
```

```
highSize = hCrtSize - lowSize;
normalSize = highSize*2;
bufferPtr = workingVector;
for ( j = 0; j < highSize; j ++ )
{
*bufferPtr = input[i][j];
bufferPtr++;
*bufferPtr = input[i][j+lowSize];
bufferPtr++;
}
if ( lowSize > highSize )
workingVector[normalSize] = input[i][highSize];
Reflection(workingVector, workingVector + hCrtSize - 1);
bufferPtr = workingVector;
for ( j = 0; j < normalSize; )
{
input[i][j++] = SynHighPass(bufferPtr);
bufferPtr++;
input[i][j++] = SynLowPass(bufferPtr);
bufferPtr++;
}
if ( lowSize > highSize )
input[i][normalSize] = SynHighPass(bufferPtr);
}
}
int imgSize = vSize*hSize;
bufferPtr=coeff, bufferEnd = coeff+imgSize;
while ( bufferPtr < bufferEnd )
{
*bufferPtr = Round(*bufferPtr);
if (*bufferPtr > maxOrig)
*bufferPtr = maxOrig;
else
if (*bufferPtr < minOrig)
*bufferPtr = minOrig;
bufferPtr ++;
}
delete floatIn;
delete hTransSize;
delete vTransSize;
delete input;
return TRUE;
}
void WaveletTransform::SP_TransformStep(int *input, int *output, int size)
{
int i, k, d1, d2;
int lowSize = (size+1)/2;
int highSize = size - lowSize;
int *l = output;
int *h = output + lowSize;
int *in = input;
```

```
for (i = k = 0; i < highSize; i++, k += 2)
{
l[i] = (in[k] + in[k+1]) >> 1;
h[i] = in[k] - in[k+1];
}
d2 = l[0] - l[1];
h[0] -= d2 >> 2;
for (i = 1; i < highSize-1; i++)
{
d1 = d2;
d2 = l[i] - l[i+1];
h[i] -= (((d1 + d2 - h[i+1]) << 1) + d2 + 3) >> 3;
}
h[i] -= d2 >> 2;
if ( lowSize > highSize )
l[highSize] = in[2*highSize];
}
void WaveletTransform::SP_InvertStep(int *input, int *output, int size)
{
int i, k, d1, d2, t;
int lowSize = (size+1) >> 1;
int highSize = size - lowSize;
int *l = input;
int *h = input + lowSize;
int *out = output;
t = (h[highSize-1] += (d1 = l[highSize-2] - l[highSize-1]) >> 2);
for (i = highSize - 2; i > 0; i--)
{
d2 = d1;
d1 = l[i-1] - l[i];
t = (h[i] += (((d1 + d2 - t) << 1) + d2 + 3) >> 3);
}
h[0] += d1 >> 2;
for (i = k = 0; i < highSize; i++, k += 2)
{
out[k] = l[i] + ((h[i] + 1) >> 1);
out[k+1] = out[k] - h[i];
}
if ( lowSize > highSize )
output[2*highSize] = l[highSize];
}
//
// Файл wavelet.h
//
#ifndef WAVELET_H
#define WAVELET_H
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <string.h>
```

```
class WaveletTransform
{
public:
WaveletTransform ();
~WaveletTransform ();
BOOL FWT (float *coeff, int trSteps, int hSize, int vSize);
BOOL IWT (float *coeff, int trSteps, int hSize, int vSize,
int maxOrig, int minOrig);
int npad;
protected:
void Reflection(float * h, float * t);
void SP_TransformStep(int *input, int *output, int size);
void SP_InvertStep(int *input, int *output, int size);
};
#endif
```