WAVELET COMPRESSION
WITH FEATURE PRESERVATION
AND DERIVATIVE DEFINITION

Oleg Kiselyov and Paul Fisher

Department of Computer Sciences
P.O.Box 13886
University of North Texas, Denton Texas 76203
Phone: 817-565-2767  Fax: 817-565-2799
Email: oleg@ponder.csci.unt.edu
FEATURES

• Loose wavelet basis (frame) *
  ⇒ Lower entropy of the wavelet decomposition

• Trimming of the Laplacian pyramid according to the "contrast-frequency ratio" *

• Run-length + Arithmetic/LZW compression of the Laplacian pyramid

→ • Introducing Criteria Sets
→ Criteria Set area: control over the amount of loss at particular locations of the picture

→ • Discrete gradient of the picture from the Wavelet decomposition

→ • A number of C++ functions dealing with images
  ⇒ image manipulation language

*As suggested by DeVore, Jawerth & Lucier, DCC 91
Unique features are marked with →
WAVELET DECOMPOSITION

\[ f(x,y) = \sum_{k=0}^{M} [P^k f - P^{k-1} f] = \sum_{k=0}^{M} \sum_{l,m} c_{lm}^k \Phi_{lm}^k (x,y) \] (1)

where

\[ f(x,y) - \text{pixel value at } x^{th} \text{ row, } y^{th} \text{ column; } x,y = 0..N-1 \]

\[ \Phi_{lm}^k (x,y) - \text{a frame (basis) function} \]

\[ = 1 \text{ over the square of size } N/2^k \]

\[ \text{with upper left corner at } (l N/2^k, m N/2^k) \] (2)

\[ = 0 \text{ everywhere else} \]

\[ M = \log N \]

\[ P^k f - \text{projector, computed as the mean intensity} \]

\[ \text{over the squares of size } N/2^k \text{ rounded to the closest integer} \]
ALGORITHM

a) Building Quadtree (Gaussian pyramid)

\[ a_{lm}^k = f(l,m); \quad k = M, \ l,m=0..N-1 \]
\[ a_{lm}^{k-1} = \text{round} \frac{1}{4} \left( a_{2l,2m}^k + a_{2l+1,2m}^k + a_{2l,2m+1}^k + a_{2l+1,2m+1}^k \right) \]
\[ k=M,M-1,...,1; \ l,m=0..2^{k-1} - 1 \]

b) Building Laplacian Pyramid

\[ c_{00}^0 = a_{00}^0 \]
\[ c_{lm}^{k+1} = a_{lm}^{k+1} - a_{l/2,m/2}^k \]
\[ k=0,1,..M-1; \ l,m=0..2^{k+1} - 1 \]

c) Trimming/Quantization

==＞ SEE THE NEXT PAGE

d) Run-Length Coding

of zero gaps left after trimming

e) Arithmetic/LZW Compressing

the entire output file

NOTE,

Time complexity of the entire algorithm \( \propto \) size of the image
TRIMMING

I. Uniform

- Sets \( c_{lm}^k = 0 \) if \( \|c_{lm}^k \Phi_{lm}^k (x,y)\| < T \), a threshold
- Keeps only **significant features of the image**
  
  \[(\text{contrast}) \times (\text{grain-level}) > \text{threshold}\]
  
  uniformly over the entire picture

→ II. Non-Uniform

- Preserves certain image features in lossy compression according to a (predefined) Criteria Set
- Criteria Set Area
  
  sets out regions to trim finer/harsher
  
  ⇒ **user-specified weight function/image** \( r(x,y) \)

- Trimming criterion
  
  \[|c_{lm}^k| \|\Phi_{lm}^k (x,y)\|_{r(x,y)} < T\]
  
  \[(\text{contrast}) \times (\text{grain-level}) \times (\text{weight}_{(x,y)}) < \text{threshold}\]

- Benefits
  
  - Areas of special interest are encoded almost lossless
  - Higher compression ratio
  - Very smooth transition between coarse/fine areas
DISCRETE GRADIENT

Definition

\[ Df(x,y) = |D_xf(x,y)| + |D_yf(x,y)| \]  
(3)

\[ D_xf(x,y) = f(x+1,y) - f(x-1,y), \]
\[ D_yf(x,y) = f(x,y+1) - f(x,y-1), \]  
(4)

Computation

\[ D_xf(x,y) = \sum_{k=0}^{M} \sum_{l,m} c_{lm}^k D_x\Phi_{lm}^k (x,y) \]

The point

- Highlighting lines, borders, etc.
- Control over the scale of non-regularities to emphasize
- Preventing the noise enhancement
EXAMPLE of PROGRAM

// This may look like C code, but it is really -*- C++ -*-
// Main module

#include "laplpyr.h"
#include <builtin.h>
extern void system(const char * command);

main()
{
    IMAGE image("../old_images/lenna.xwd");
    image.display("Original image");
    LaplPyr lp(image,log2(image.q_nrows())+1);
    #if 0
        IMAGE weight(image);       // Non-uniform Trimming
        weight = 1;
        weight.square_of(256,rowcol(127,127)) = 10;
        lp.parse_coeffs(400,weight);
    #else
        lp.parse_coeffs(400);      // Uniform Trimming
    #endif
    lp.write("/tmp/aa");
    system("comp_ratio ../old_images/lenna.xwd /tmp/aa");
    LaplPyr lp1("/tmp/aa");     // Read the pyramid back

    IMAGE & new_image = *(lp1.compose());
    new_image.display("Reconstructed image");
    compare(image,new_image,"Original and reconstructed images");

    IMAGE diff_image(image);
    diff_image = image; diff_image -= new_image;
    message("-->Root mean square error is %g", 
            sqrt( (diff_image * diff_image) / diff_image.q_nrows() / 
                  diff_image.q_ncols() ));

    IMAGE & d_image = *(lp1.derivative_image());
    d_image *= 4;
    d_image.invert();
    d_image.display("Derivative image");
}
Original $512 \times 512 \times 8$ image "lenna"

Discrete Derivative

Compression 12:1, threshold 50

Discrete Derivative

Compression 58:1, threshold 400

Discrete Derivative

Discrete Derivative

Non-uniform trimming

threshold 400, weight function:

Compression 21:1