MegaWave `http://megawave.cmla.ens-cachan.fr`

Jacques Froment, LMBA UMR 6205, Université de Bretagne Sud

Basic concepts

MegaWave is a software environment designed to help people to write algorithms on signal processing and image analysis. This means that the programmer should focus his mind on the algorithm only and not on pure computer problems (such as how to read an image, how to display a movie on the screen, how to select a curve with the mouse, how to manage an object library,...)

The solution adopted by MegaWave is the concept of modules, that is, of black boxes where the algorithms are described as C functions, without any assumption about the context in which the modules may be executed. It is the task of the MegaWave compiler (basically a MegaWave preprocessor followed by a standard C compiler) to make from a module a run-time command, a function of a library or a function to be run under a window-oriented interpreter. MegaWave also automatically generates the documentation associated with each module so that only the mathematical or algorithmic description needs to be written.

At last but not least and besides basic modules, MegaWave offers a collection of modules associated with publications in image processing, enabling reproducible research.

History

MegaWave has a long history, starting in the late 80's with MegaWave1, which essentially was a set of C files related to the wavelet transform. MegaWave2 was born in 1993 at the CEREMADE, a mathematical laboratory of the Paris-IX Dauphine University, France, inside a research group in image processing conducted by Jean-Michel Morel. The first version has been publicly released in 1994.

Version 1.x of MegaWave2 required users to ask for a (free) license, by filling a registration form. The kernel (system library and compiler) was not open and was pre-compiled for different machine architectures. The modules were open source. MegaWave2 became completely open in 2000 starting with version 2.00. This version and the following have been supported by the CMLA, ENS Cachan, as a consequence of the move of the Morel's team to this lab.

Last officially released version is getting old (June, 2007), but works are still in progress.

People

As MegaWave1, MegaWave2 has been created by *Jacques Froment* under the supervision of *Jean-Michel Morel*, but with the help of a greater number of people. Since 1998 *Lionel Moisan* is in charge of collecting and updating public modules and user's macros. He also became the main contributor in research modules. In total more than 30 researchers contributed in MegaWave2 by writing modules.

Through the years, MegaWave has been used by more than one hundred universities and public or private research centers. In addition to the research aspect, MegaWave is also useful in some universities for teaching at Masters level.

Future

MegaWave must continue to evolve, to reflect the expectations of its users. For example the free license, the C code and installation tools should be adapted to the standards of free software. *Nico-*

las Limare started a work in this direction, which should be continued. I believe than MegaWave is complementary to the project IPOL. By being a on line journal, IPOL offers the best answer to reproducible research. For its part, MegaWave can offer an pleasant environment to generate standard source codes for IPOL.

Content

The last public MegaWave2 package comes with 350 fully documented modules. Here are the main ones, with the name of the authors (who may differ from the original authors of the algorithm):

[**Fourier and wavelet transforms**]

- FFT in 1D and 2D (Lionel Moisan)
- Image rotation and translation using Fourier interpolation (Pascal Monasse)
- Image zooming by zero-padding (Lionel Moisan)
- Phase randomization (Lionel Moisan)
- Wiener filtering (Lionel Moisan)
- 1D and 2D orthogonal and biorthogonal wavelet transforms (Jean-Pierre D'Alès)
- 2D wavelet packets decomposition (François Malgouyres)
- 1D wavelet transform using Stark's algorithm (Claire Jonchery, Amandine Robin)
- Ridgelet transform of an image (Claire Jonchery, Amandine Robin)

[**Curves and level lines representation and applications**]

- Image matching based on level lines (José-Luis Lisani, Pablo Muse, Frédéric Sur)
- Geometric Affine Scale Space of Curves (Lionel Moisan)
- Generalized Curve Shortening Flow of a curve (Frédéric Cao, Lionel Moisan)
- Detect meaningful alignments in an image (Lionel Moisan)
- Extract local or nonlocal meaningful contrasted level lines (Lionel Moisan, Frédéric Cao)
- Extract maximal meaningful edges (Lionel Moisan)
- Fast Level Sets Transform (Pascal Monasse)
- Grain filter (Pascal Monasse, Frédéric Guichard)
- Compute T and X junctions (Vicent Caselles, Bartomeu Coll, Jacques Froment, José-Luis Lisani)
- Image disocclusion (Simon Masnou)
- Level lines image interpolation using the AMLE model (absolutely minimizing Lipschitz interpolant) with an implicit Euler scheme (Jean Pierre D'Alès, Jacques Froment, Catalina Sbert)
- Level lines image interpolation using AMLE, 3D case and inf-sup scheme (Frédéric Cao)

[**Restoration**]

- Image denoising by Total Variation minimization using a relaxation algorithm (Antonin Chambolle)
- Image denoising by Total Variation minimization using Chambolle's dual algorithm (Lionel Moisan)
- Image denoising by Total Variation minimization using Rudin-Osher-Fatemi algorithm (Lionel Moisan)
- Image deblurring by Total Variation minimization (Lionel Moisan)
- Image denoising by means of ridgelet thresholding (Claire Jonchery, Amandine Robin)
- Restoration of signals using total variation and thresholding in orthonormal bases (Jacques Froment)

- Non-Local Means image denoising (Lionel Moisan)

[**Other filterings**]

- Affine Morphological Scale Space and Mean Curvature Motion (Frédéric Guichard, Lionel Moisan)
- Affine Morphological Scale Space as a stack filter (Lionel Moisan)
- Rudin Shock filter (Lionel Moisan)
- Mathematical Morphology operators : erosion, dilation, opening, closing, median, Lipschitz inf or sup envelope (Lionel Moisan)
- Heat equation (Lionel Moisan)
- Inf-Sup scheme and median-median filtering (Frédéric Guichard, Denis Pasquignon)
- Image sharpen by selection of parallel level lines (Pascal Monasse, Frédéric Guichard)
- Multiscale Analysis of Movies (Frédéric Guichard, Lionel Moisan)

[**Other feature detection**]

- Canny-Deriche's Edge Detector (Yann Guyonvarc'h)
- Harris corner detector (Frédéric Cao)
- Detect maximal meaningful vanishing points/regions (Andres Almansa)
- Image skeleton using a PDE (Denis Pasquignon)
- Shape recognition (Thierry Cohignac, Lionel Moisan)

[**Compression**]

- Wavelet compression via EZW algorithm (Jean-Pierre D'Alès)
- Vector quantization via LBG algorithm (Jean-Pierre D'Alès)
- Vector quantization in the wavelet domain (Jean-Pierre D'Alès)

[**Transformations on the image's domain**]

- Image interpolations (Lionel Moisan)
- Affine or projective mapping (Lionel Moisan)
- Image zooming, translation and rotation (Lionel Moisan)

[**Segmentation and snakes**]

- Region-growing method using the energy of Mumford and Shah (Georges Koepfler)
- Texture segmentation using the energy of Mumford and Shah (Yann Guyonvarc'h)
- Level sets implementation of the snakes model (Françoise Dibos, Kamal Lakhiari)
- Kimmel and Bruckstein snake model for contour detection (Lionel Moisan)

[**Motion**]

- Compute optical flow using Horn and Schunck method (Olivia Sanchez)
- Motion segmentation using the Aubert-Deriche-Kornprobst method (Toni Buades)
- Compute optical flow using Weickert and Schnörr method (Florent Ranchin)

Implementation

MegaWave2 is written using the C language, Kernighan & Ritchie version (for historical reasons) and with Bourne shell scripts, using standard Unix commands. Modules are written in the same C language (an ANSI C version is on the road), with an encapsulated header giving additional information to the preprocessor (such as input/output, options, ...) so that one single file (the "module.c" file) is enough to generate the run-time command and the documentation skeleton. One can also write modules in Bourne shell script (e.g. to run a demo). Last version of MegaWave2 runs on Linux ix86 architecture. Previous versions were running on HP-UX, Solaris and SGI IRIX, so you may expect the system to work on a large variety of Unix distributions with little additional effort. Parallel programming is available via the OpenMP library, although no released module uses it at that time.

Example

This example shows the source of the module `fftconvol` which computes the convolution of an input gray level image `in` with a real kernel `filter` in the Fourier domain, following the formula

$$\widehat{\text{out}}(i) = \widehat{\text{in}}(i) \times \text{filter}(i)$$

```
/*---------------------------- MegaWave2 Module -----------------------------*/
/* mwmodule
 name = {fftconvol};
 version = {"1.2"};
 author = {"Lionel Moisan"};
 function = {"2D Fourier-convolution of a fimage"};
 usage = {
    in->in           "input Fimage",
    filter->filter   "convolution filter in Fourier domain (Fimage)",
    out<-out         "output Fimage"
};
-----------------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
#include "mw.h"

extern void fft2d(); // Use this module

Fimage fftconvol(in,filter,out)
     Fimage in,filter,out;
{
  int i,nx,ny;
  Fimage re,im;

  nx = in->ncol; ny = in->nrow; // Size of the input image

  if (filter->ncol!=nx || filter->nrow!=ny) // Check image and filter size
    mwerror(USAGE,1,"Input image and filter dimensions do not match !\n");

  re = mw_new_fimage();  im = mw_new_fimage(); // Create images <re> and <im>

  fft2d(in,NULL,re,im,0); // Compute the 2D Fourier transform

  for (i=nx*ny;i--;) { // Multiplication in the Fourier domain
    re->gray[i] *= filter->gray[i];
    im->gray[i] *= filter->gray[i];  }

  out = mw_change_fimage(out,ny,nx); // Allocate memory for the output image
  if (!out) mwerror(FATAL,1,"Not enough memory\n"); // If allocation failed

  fft2d(re,im,out,NULL,1); // Inverse 2D Fourier transform

  mw_delete_fimage(re); mw_delete_fimage(im); // Free memory <re> and <im>

  return(out); // Return the computed image (optional)
}
```