The Npic library

Tutorial

Edouard Thiel

26 november 2008

Contents

Chapter 1

Introduction

Npic is a library, that is to say, a set of types and functions, which allows you to write image analysis programs in C language.

The library provides a type `Npic_image` for handling bitmap images in memory. Choice is given for dimension (2 to 6), pixels type (char, long, double, 4 shorts), pixel access (linear or cartesian coordinates) ; an external border can be added for simplifying programs working with masks (distance transforms, convolutions, etc).

Npic provides the NPZ compressed file format (extension `.npz`) which allows to store any `Npic_image` to disk. Functions are also given for other formats, such as the 3D VOL format or the 2D PGM format; the files can be read or write directly gzip compressed, which save a lot of disk space! A set of command-line tools are given in directory tools/, which you can call in shell scripts.

Note: this document is not finished, it will be improved a.s.a.p.

Chapter 2

Getting started

2.1 First program

```
#include <npic.h>

int main ()
{
    int x, y;

    Npic_image *np1;
    Npic_image_2l *p1;

    /* Create a 2L image = 2D image with signed long pixels (4 bytes). height
       is ymax=12, width is xmax=20, external borders are ybor=0, xbor=0. */
    np1 = NpicCreateImage_2l (12, 20, 0, 0);
    if (np1 == NULL) exit (1);

    /* Assign values to pixels */
    p1 = NpicCastImage (np1);
    for (y = 0; y < p1->ymax; y++)
    for (x = 0; x < p1->xmax; x++)
        p1->pix[y][x] = (x + y) % 16;

    /* Print image */
    NpicPrintImage (np1);

    /* Destroy image */
    NpicDestroyImage (np1);
    exit(0);
}
```

2.2 Compiling

In this section, we show four different ways for compiling a program using libNpic : on command line, with a shell script, with a basic Makefile, with a more sophisticated one.

For compiling the previous example `demo1.c`, type :

```
gcc demo1.c -o demo1 `/npic_path/npic-cfg --cflags --libs`
```

The command between backquotes ‘ ‘ allows to recover the options which depend from Npic installation; you just have to replace /npic_path with the actual absolute path.

Another way for compiling is to use the following sh script :

```
#! /bin/sh
p=/npic_path            # replace with actual path
f=‘basename $1 .c‘
gcc $f.c -o $f ‘$p/npic-cfg --cflags --libs‘
```

Call this script "ncomp", save, type "chmod +x ncomp". For compiling a file "ex.c", type "./ncomp ex.c" or "./ncomp ex". For compiling a file and run the program if the compilation succeed, type "./ncomp ex.c && ex".

Now here is a basic Makefile; one includes the Npic configuration file /npic_path/.config, where the usefull variables are declared, in particular, $(CC), $(NPIC_CFLAGS) and $(NPIC_LIBS).
Beware, be sure that the indented lines start with a [TAB].

```
include /npic_path/.config          # replace with actual path

.c.o :
    $(CC) -c $(NPIC_CFLAGS) $*.c

ex1 : ex1.o
    $(CC) -o $@ $@.o $(NPIC_LIBS)
```

Finally, the following Makefile allows to compile several programs and to clean up the directory.

```
include /npic_path/.config          # replace with actual path

.c.o :
    $(CC) -c $(NPIC_CFLAGS) $*.c

# Add here the name of your programs
EXECS = ex1 ex2 ex3

help ::
    @echo "Options of make : help all clean distclean $(EXECS)"

# Add here the dependencies
ex1 : ex1.o
ex2 : ex2.o
ex3 : ex3.o

all :: $(EXECS)

$(EXECS) :
```

```
    $(CC) -o $@ $@.o $(NPIC_LIBS)

clean ::
    \rm -f *.o core

distclean :: clean
    \rm -f $(EXECS)
```

2.3 Function receiving a Npic_image

```
#include <npic.h>

void my_draw (Npic_image *np1)
{
    if (NpicImageIsOK (np1, __func__) != NPIC_SUCCESS)
        return;

    switch (np1->type) {

        case NPIC_IMAGE_2L : {
            Npic_image_2l *p1 = NpicCastImage (np1);
            int x, y;

            for (y = 0; y < p1->ymax; y++)
            for (x = 0; x < p1->xmax; x++)
                p1->pix[y][x] = (x + y) % 16;
            return;
        }

        case NPIC_IMAGE_3L : {
            Npic_image_3l *p1 = NpicCastImage (np1);
            int x, y, z;

            for (z = 0; z < p1->zmax; z++)
            for (y = 0; y < p1->ymax; y++)
            for (x = 0; x < p1->xmax; x++)
                p1->pix[z][y][x] = (x + y + z) % 16;
            return;
        }

        default :
            np1->gen.ok = NpicError (__func__, NPIC_ERR_UNEX_NPIC, "");
    }
}

int main ()
{
    Npic_image *np1;

    /* Create a 2L image = 2D image with signed long pixels (4 bytes). height
       is ymax=12, width is xmax=20, external borders are ybor=0, xbor=0. */
    np1 = NpicCreateImage_2l (12, 20, 0, 0);
    if (np1 == NULL) exit (1);

    /* Assign values to pixels */
    my_draw (np1);

    /* Print image */
    NpicPrintImage (np1);
```

```
    /* Destroy image */
    NpicDestroyImage (np1);
    exit(0);
}
```

2.4 Function receiving several Npic_image

```
#include <npic.h>

void my_draw (Npic_image *np2, Npic_image *np1, double c1, double c2)
{
    if (NpicImageIsOK_DS1 (np2, np1, __func__) != NPIC_SUCCESS)
        return;

    if (NpicSameImage (np2, np1, NPIC_TYPE | NPIC_SIZE) != NPIC_SUCCESS) {
        np2->gen.ok = NpicError (__func__, NPIC_ERR_INCOMPAT, "");
        return;
    }

    switch (np2->type) {

        case NPIC_IMAGE_2L : {
            Npic_image_2l *p2 = NpicCastImage (np2),
                          *p1 = NpicCastImage (np1);
            int x, y;

            for (y = 0; y < p2->ymax; y++)
            for (x = 0; x < p2->xmax; x++)
                p2->pix[y][x] = p1->pix[y][x]*c1 + c2;

            return;
        }

        case NPIC_IMAGE_3L : {
            Npic_image_3l *p2 = NpicCastImage (np2),
                          *p1 = NpicCastImage (np1);
            int x, y, z;

            for (z = 0; z < p2->zmax; z++)
            for (y = 0; y < p2->ymax; y++)
            for (x = 0; x < p2->xmax; x++)
                p2->pix[z][y][x] = p1->pix[z][y][x]*c1 + c2;

            return;
        }

        default :
            np2->gen.ok = NpicError (__func__, NPIC_ERR_UNEX_NPIC, "");
    }
}

int main ()
{
    Npic_image *np1, *np2;
    Npic_image_2l *p1;
    int x, y;

    /* Create a 2L image = 2D image with signed long pixels (4 bytes). height
       is ymax=12, width is xmax=20, external borders are ybor=0, xbor=0. */
    np1 = NpicCreateImage_2l (12, 20, 0, 0);
    if (np1 == NULL) exit (1);
```

```
    /* Assign values to pixels */
    p1 = NpicCastImage (np1);
    for (y = 0; y < p1->ymax; y++)
    for (x = 0; x < p1->xmax; x++)
        p1->pix[y][x] = (x + y) % 16;

    /* Create a second image with same size and type than np1 */
    np2 = NpicDupImage (np1);
    if (np2 == NULL) exit (1);

    /* Compute np2 pixels values */
    my_draw (np2, np1, 2, 4);

    /* Print image */
    NpicPrintImage (np2);

    /* Destroy image */
    NpicDestroyImage (np1);
    NpicDestroyImage (np2);
    exit(0);
}
```