The Npic library

Reference Manual

Edouard Thiel

9 december 2020

Contents

Chapter 1

Npic Functions

In this chapter we describe all public functions from libNpic. In each section we give the links to the sources, which are self-documented.

In the following, a function is said *verbose* if it prints something in the console, else it is said *silent*.

Each object (image, mask, etc) has a field telling if it is *ok* or not. When created, an object has ok set to true ; any failing operation will set ok to false. From the moment that an object is not ok, all following operations will immediatly return, without doing nothing (the object can just be destroyed).

1.1 Images in memory

1.1.1 Creation of an image

Sources: image_creat.h , image_creat.c .

• **NpicCreateImage_***

```
Npic_image *NpicCreateImage_2c (int ymax, int xmax,
                                int ybor, int xbor);
Npic_image *NpicCreateImage_2l (int ymax, int xmax,
                                int ybor, int xbor);
Npic_image *NpicCreateImage_2d (int ymax, int xmax,
                                int ybor, int xbor);
Npic_image *NpicCreateImage_2q (int ymax, int xmax,
                                int ybor, int xbor);
Npic_image *NpicCreateImage_3c (int zmax, int ymax, int xmax,
                                int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_3l (int zmax, int ymax, int xmax,
                                int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_3d (int zmax, int ymax, int xmax,
                                int zbor, int ybor, int xbor);
```

```
Npic_image *NpicCreateImage_3q (int zmax, int ymax, int xmax,
                                int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_4c (int tmax, int zmax, int ymax, int xmax,
                                int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_4l (int tmax, int zmax, int ymax, int xmax,
                                int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_4d (int tmax, int zmax, int ymax, int xmax,
                                int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_4q (int tmax, int zmax, int ymax, int xmax,
                                int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_5c
                      (int smax, int tmax, int zmax, int ymax, int xmax,
                       int sbor, int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_5l
                      (int smax, int tmax, int zmax, int ymax, int xmax,
                       int sbor, int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_5d
                      (int smax, int tmax, int zmax, int ymax, int xmax,
                       int sbor, int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_5q
                      (int smax, int tmax, int zmax, int ymax, int xmax,
                       int sbor, int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_6c
           (int rmax, int smax, int tmax, int zmax, int ymax, int xmax,
            int rbor, int sbor, int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_6l
           (int rmax, int smax, int tmax, int zmax, int ymax, int xmax,
            int rbor, int sbor, int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_6d
           (int rmax, int smax, int tmax, int zmax, int ymax, int xmax,
            int rbor, int sbor, int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicCreateImage_6q
           (int rmax, int smax, int tmax, int zmax, int ymax, int xmax,
            int rbor, int sbor, int tbor, int zbor, int ybor, int xbor);
```

Full creation of an image. All pixels are set to 0, including border pixels.

Return the image on success, else `NULL`. Verbose.

The suffix of `NpicCreateImage_`*dt* gives the dimension *d* (2, 3, 4, 5 or 6) and the pixels type *t* (`Npic_c`, `Npic_l`, `Npic_d`, `Npic_q`):

- ▷ c stands for unsigned char (8 bits integer);
- ▷ l stands for signed long (32 bits integer);
- ▷ d stands for double (floating 64 bits number);
- ▷ q stands for quadri signed short (four 16 bits integer).

Each image can have an "external border", different for each coordinate. This kind of

feature is especially useful with masks. The principle is the same for all coordinates, so we just explain for `x`: the interior of the image is `[0 .. xmax[`, the left border is `[-xbor .. 0[`, the right border is `[xmax .. xmax+xbor[`, the whole image is `[-xbor .. xmax+xbor[` . For no border, set `xbor` to 0.

Usage:

```
Npic_image *np;
np = NpicCreateImage_.. (..., ymax, xmax, ..., 0, 0, ...);
if (np == NULL) exit (1);
....
NpicDestroyImage (np);
```

• **NpicCreateImageNS**

```
Npic_image *NpicCreateImageNS (int ntype, Npic_vec *size);
```

Creation of an image without borders, from image `ntype` and `size`, where `ntype` is one of the constants:
`NPIC_IMAGE_2C`, `NPIC_IMAGE_2L`, `NPIC_IMAGE_2D`, `NPIC_IMAGE_2Q`, `NPIC_IMAGE_3C`, `NPIC_IMAGE_3L`, `NPIC_IMAGE_3D`, `NPIC_IMAGE_3Q`, `NPIC_IMAGE_4C`, `NPIC_IMAGE_4L`, `NPIC_IMAGE_4D`, `NPIC_IMAGE_4Q`, `NPIC_IMAGE_5C`, `NPIC_IMAGE_5L`, `NPIC_IMAGE_5D`, `NPIC_IMAGE_5Q`, `NPIC_IMAGE_6C`, `NPIC_IMAGE_6L`, `NPIC_IMAGE_6D`, `NPIC_IMAGE_6Q`.

All pixels are set to 0. Return image on success, else return NULL. Verbose.

Usage:

```
Npic_image *np;
Npic_vec vs; vs.x = 100; vs.y = 200; vs.z = 150;

np = NpicCreateImageNS (NPIC_IMAGE_3C, &vs);
if (np == NULL) ...
....
NpicDestroyImage (np);
```

• **NpicCreateImageDPS**

```
Npic_image *NpicCreateImageDPS (int dim, int pixeltype, Npic_vec *size);
```

Creation of an image without borders, from dimension `dim` (in 2..6), `pixeltype` (`NPIC_C`, `NPIC_L`, `NPIC_D` or `NPIC_Q`), and `size`. All pixels are set to 0.
Return image on success, else return NULL. Verbose.

Usage:

```
Npic_image *np;
Npic_vec vs; vs.x = 100; vs.y = 200; vs.z = 150;

np = NpicCreateImageDPS (3, NPIC_C, &vs);
```

```
if (np == NULL) ...
....
NpicDestroyImage (np);
```

• **NpicDestroyImage**

```
void NpicDestroyImage (Npic_image *np);
```

Free memory of an image (even if NULL or not ok). Silent.

1.1.2 Duplication of an image

Sources: image_creat.h , image_creat.c .

• **NpicDupImage**

```
Npic_image *NpicDupImage (Npic_image *src);
```

Create an image `dest` of same type, size and border than `src`. The pixels are *not* copied: all pixels of `dest` are set to 0, including border pixels. Do nothing if `src` is not ok.

Return `dest` on success, else `NULL`. Verbose.

• **NpicDupImageB**

```
Npic_image *NpicDupImageB (Npic_image *src,
                           int tbor, int zbor, int ybor, int xbor);
Npic_image *NpicDupImageB6 (Npic_image *src,
       int rbor, int sbor, int tbor, int zbor, int ybor, int xbor);
```

The same with other border sizes. Depending on dimension, `rbor`, `sbor`, `tbor` or `zbor` are ignored.

• **NpicSwapImage**

```
void NpicSwapImage (Npic_image *np1, Npic_image *np2);
```

Swap all fields between images `np1` and `np2` (even if not ok). Silent.

1.1.3 Checks on images

Sources: image_check.h , image_check.c .

• **NpicImageIsOK**

```
int NpicImageIsOK (Npic_image *np, const char *funcname);
```

Check if image is not NULL, has a correct type and is ok. On error, print an error message containing funcname.

Return NPIC_SUCCESS on success, else error code.

• **NpicImageIsOK_DS***

```
int NpicImageIsOK_DS1 (Npic_image *dest, Npic_image *src1,
                       char const *funcname);
int NpicImageIsOK_DS2 (Npic_image *dest, Npic_image *src1,
                       Npic_image *src2, const char *funcname);
int NpicImageIsOK_DS3 (Npic_image *dest, Npic_image *src1,
                       Npic_image *src2, Npic_image *src3,
                       const char *funcname);
```

Check if all images are ok. On error, set not ok for dest, and print an error message containing funcname.

Return NPIC_SUCCESS on success, else error code.

• **NpicSameImage**

```
int NpicSameImage (Npic_image *np1, Npic_image *np2, int what);
```

Test if the two images have same type, dimension, size or border (even if they are not ok). what is a binary OR between NPIC_TYPE, NPIC_DIM, NPIC_SIZE, NPIC_BORD.

Return NPIC_SUCCESS on success, else error code. Silent.

• **NpicImageIsSquare**

```
int NpicImageIsSquare (Npic_image *np1, const char *funcname);
```

Test if the image is square. On error, set not ok for dest, and print an error message containing funcname.

Return NPIC_SUCCESS on success, else error code.

1.1.4 Type to name and vice-versa

Sources: image_ntype.h , image_ntype.c .

• **NpicImageNtype**

```
int NpicImageNtype (int dim, int pixeltype);
```

Describe the image type corresponding to dim and pixeltype, where dim is in 2..6 and pixeltype is one of NPIC_C, NPIC_L, NPIC_D or NPIC_Q.

Return one of these constants: NPIC_IMAGE_2C, .. NPIC_IMAGE_6Q, else NPIC_NONE on error. Silent.

• **NpicImagePixelType**

```
int NpicImagePixelType (int ntype)
```

Describe the pixel type from the image type `ntype`, where `ntype` is one of `NPIC_IMAGE_2C` .. `NPIC_IMAGE_6Q`.
Return one of these constants: `NPIC_C`, `NPIC_L`, `NPIC_D` or `NPIC_Q`, else `NPIC_NONE` on error. Silent.

• **NpicImageTypeName**

```
const char *NpicImageTypeName (int ntype);
```

Return a static string containing the name of the image type, such as `"NPIC_IMAGE_2C"`, else `"ERROR"`. `ntype` is one of `NPIC_IMAGE_2C` .. `NPIC_IMAGE_6Q`. Silent.

1.1.5 Accessing to the pixels

Sources: image_creat.h , image_creat.c .

• **NpicCastImage**

```
void *NpicCastImage (Npic_image *np1);
```

Return the typed image inside the n-image `np1` (even if not ok). Silent.

We call *n-image* a variable of type `Npic_image`, which can store an image having any of the allowed dimensions and pixel types. A *typed image* is a variable of type `Npic_image_`*dt* having a determined dimension *d* and pixel type *t*.

The pixels cannot be accessed in a n-image. Since `Npic_image` is a union of structs, we normally should write `Npic_image_`*dt*`*p1 = &np1->im_`*dt* to get the typed image `p1` from `np1`; thanks to this function, we can simplier write `Npic_image_`*dt*`*p1 = NpicCastImage (np1)`. How does it work ? This is actually a simple cast of the `Npic_image*` type to `void*`.

Of course, `NpicCastImage` must be called *after* successful creation of an image:

```
Npic_image   *np1;   /* the n-image     */
Npic_image_2c *p1;   /* the typed image */
...
np1 = NpicCreateImage_2c (ymax, xmax, 0, 0);
if (np1 == NULL) return;

p1 = NpicCastImage (np1);
for (y = 0; y < p1->ymax; y++)
for (x = 0; x < p1->xmax; x++)
    p1->tab[y][x] = x+y;

...
NpicDestroyImage (np1);
```

1.1.6 Copying or converting pixels

Sources: image_copy.h , image_copy.c .

• **NpicCopyImageI**

```
void NpicCopyImageI (Npic_image *dest, Npic_image *src);
```

Copy the value of each *interior* pixel from `dest` to the corresponding pixel in `src`. The two images must have same dimension and size.

Do not copy nor modify border pixels. Do nothing if `dest` or `src` is not ok. set not ok for `dest` on error. Verbose.

This function is also usefull for conversion between 2 different images types. Values can be truncated during conversion; no warning is done.

Usage:

```
Npic_image *src, *dest;

src = NpicReadImage ("file.npz");
...
dest = NpicDupImage (src);
NpicCopyImageI (dest, src);
```

• **NpicCopyImageIB**

```
void NpicCopyImageIB (Npic_image *dest, Npic_image *src);
```

The same, including border pixels. The two images must also have same border widths.

• **NpicCopyImageB**

```
void NpicCopyImageB (Npic_image *dest, Npic_image *src);
```

The same, only on border pixels. The two images must have same border.

• **NpicConvertImage_***

```
void NpicConvertImage_c (Npic_image *np1);
void NpicConvertImage_l (Npic_image *np1);
void NpicConvertImage_d (Npic_image *np1);
```

Convert image pixels to `Npic_c`, `Npic_l` or `Npic_d` format (border pixels excluded).

Do nothing if `np1` is not ok, or if pixels already have the expected type. Set not ok for `np1` on error. Verbose.

Values can be truncated during conversion; no warning is done.

1.1.7 Cutting in slices

Sources: image_slice.h , image_slice.c .

• **NpicCreateSlice_***

```
Npic_image *NpicCreateSlice_r (Npic_image *np1, int t);
Npic_image *NpicCreateSlice_s (Npic_image *np1, int t);
Npic_image *NpicCreateSlice_t (Npic_image *np1, int t);
Npic_image *NpicCreateSlice_z (Npic_image *np1, int z);
Npic_image *NpicCreateSlice_y (Npic_image *np1, int y);
Npic_image *NpicCreateSlice_x (Npic_image *np1, int x);
```

Create an image of dimension-1, containing the slice for the coordinate.

Return new image or NULL.

1.2 More on image border

Images can be created with an external border or not, see section 1.1.1 . This border can be added, changed or canceled after image creation.

1.2.1 Border size

Sources: image_border.h , image_border.c .

• **NpicSetBorderWidth**

```
void NpicSetBorderWidth (Npic_image *np,
                         int tbor, int zbor, int ybor, int xbor);
void NpicSetBorderWidth6 (Npic_image *np,
     int rbor, int sbor, int tbor, int zbor, int ybor, int xbor);
```

Change border width of image `np`. The image pixels are unchanged; the border pixels are set to 0. Depending on dimension, `rbor`, `sbor`, `tbor` or `zbor` are ignored.

Do nothing if `np` is not ok. set not ok on error. Verbose.

• **NpicSetBorderWidthSame**

```
void NpicSetBorderWidthSame (Npic_image *np, int bord);
```

Change all border widths of `np` to same value `bord`. The image pixels are unchanged; the border pixels are set to 0.

Do nothing if `np` is not ok. set not ok on error. Verbose.

• **NpicSetBorderWidthMin**

```
void NpicSetBorderWidthMin (Npic_image *np, int bord);
```

Change the border widths of `np` which are smaller than `bord` to the value `bord`. The image pixels are unchanged; the border pixels are set to 0.

Do nothing if `np` is not ok. set not ok on error. Verbose.

1.2.2 Setting border pixel values

Sources: image_border.h , image_border.c .

Depending on image type, use *function*`_i` for integer pixels, *function*`_d` for double pixels, *function*`_q` for 4 short pixels.

• **NpicFillWholeZero**

```
void NpicFillWholeZero (Npic_image *np);
```

Set each pixel to zero, including border pixels.

Do nothing if `np` is not ok. set not ok on error. Verbose.

• **NpicFillWhole_***

```
void NpicFillWhole_i (Npic_image *np, int val1);

void NpicFillWhole_d (Npic_image *np, double val1);

void NpicFillWhole_q (Npic_image *np, int a, int b, int c, int d);
```

Set all pixels to the value `val1`, including border pixels.

Do nothing if `np` is not ok. set not ok on error. Verbose.

• **NpicFillBorder_***

```
void NpicFillBorder_i (Npic_image *np, int val1);

void NpicFillBorder_d (Npic_image *np, double val1);

void NpicFillBorder_q (Npic_image *np, int a, int b, int c, int d);
```

Set border pixels to the value `val1`. The image pixels are unchanged.

Do nothing if `np` is not ok. set not ok on error. Verbose.

1.2.3 Mirroring

Sources: image_border.h , image_border.c .

• **NpicMirrorBorder**

```
void NpicMirrorBorder (Npic_image *np);
```

Fill the border with image pixels. The image pixels are unchanged. The Line 1 is copied on line -1, line 2 on line -2, etc; same for rows, etc. Useful for linear filters.

Do nothing if `np` is not ok. set not ok on error. Verbose.

As an example, see test-mirror.c.

1.3 Image Files

Npic knows a number of image formats, which are listed in this section.

1.3.1 By extension

Sources: files_byext.h , files_byext.c , files_hints.h , files_hints.c , files_gz.h , files_gz.c .

The easiest way to read or write a file is to let Npic guess the format from the file extension. If the extension ends by ".gz", ".bz2" or ".7z", then the file is (un)compressed on fly.

Some file formats allow *hints* at the end of the file name (currently, formats: pnm, pbm, pgm, ppm). A hint is a list of comma-separated words after a ":" and without blanks. The hints are memorized and suppressed from the name. For instance in a file named `"tmp1.pnm.gz:16-bit,little-endian"`, the real name is `"tmp1.pnm.gz"` and the hints are `"16-bit"` and `"little-endian"`.

• **NpicWriteImage**

```
int NpicWriteImage (Npic_image *np, const char *filename);
```

Write image in file according to its extension. Do nothing if image is not ok.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

• **NpicReadImage**

```
Npic_image *NpicReadImage (const char *filename);
```

Read image from file according to its extension and create appropriate image.

Return image on success, else return `NULL`. Verbose.

• **NpicInfoImage**

```
int NpicInfoImage (const char *filename);
```

Print image informations according to its extension.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

Usage:

```
Npic_image *np;
```

```
np = NpicReadImage ("image1.pgm.gz");
if (np == NULL) exit (1);

if (NpicWriteImage (np, "image2.npz") != NPIC_SUCCESS) exit (1);

NpicDestroy (np);
```

As an example, see npic-conv.c which converts any image file format in another (if possible), and npic-info.c which prints informations of image files.

1.3.2 Npic compressed format NPZ

Sources: files_npz.h , files_npz.c .

The NPZ file format (with `.npz` extension) is the proposed format to read and save Npic images, whatever the dimension, the pixel type and the endianness.

The beginning of the file is a text header describing the content (use Unix command `head` to watch it) and the text properties, the next is the bitmap compressed with gzip. See section 3.1 for specifications.

The NPZ file format can store text properties; they are not limited in number neither in size, see section 1.7.3

• **NpicWriteNPZ**

```
int NpicWriteNPZ (Npic_image *np, const char *filename);
```

Write file in NPZ format. Only interior pixels are saved. Do nothing if image is not ok.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

• **NpicReadNPZ**

```
Npic_image *NpicReadNPZ (const char *filename);
```

Read file in NPZ format.

Return image on success, else return `NULL`. Verbose.

• **NpicInfoNPZ**

```
int NpicInfoNPZ (const char *filename);
```

Print infos of a file in NPZ format.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

1.3.3 PAN format for images and volumes

Sources: files_pan.h , files_pan.c .

The PAN file format (extension `.pan`) is a file format for 2D/3D/3D+t images, proposed by Regis Clouard and others from Greyc, in the Pandore library, and written in C++.

The PAN header is a binary struct, followed by attributes; the datas are raw. The format doesn't allow text properties. The PAN file format can also store objects as graphs, but this is not implemented in Npic.

For non compressed files (with extension `.pan`), set `comp = NPIC_COMPRESS_NONE`. You can directly read or write compressed files, having extension `.pan.gz`, `.pan.bz2` or `.pan.7z`, by setting `comp = NPIC_COMPRESS_GZ`, `NPIC_COMPRESS_BZ2` or `NPIC_COMPRESS_7Z`.

• **NpicWritePAN**

```
int NpicWritePAN (Npic_image *np, const char *filename, Npic_file_compress comp);
```

Write file in PAN format and save Npic_image image, according to its type.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

• **NpicReadPAN**

```
Npic_image *NpicReadPAN (const char *filename, Npic_file_compress comp);
```

Read file in PAN format and create Npic_image image of corresponding type.

Return image on success, else return `NULL`. Verbose.

• **NpicInfoPAN**

```
int NpicInfoPAN (const char *filename, Npic_file_compress comp);
```

Print infos of a file in PAN format.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

1.3.4 VOL format for volumes

Sources: files_vol.h , files_vol.c .

The VOL file format (extension `.vol`) is a file format for 3D images, proposed by David Coeurjolly and Alexis Guillaume, in the SimpleVol library, written in C++.

The VOL header is a list of text properties, describing the dimensions, the endianness, etc, some of these being mandatory. The total number of properties is limited to 64 and their length to 128 char each. See section 1.7.3

For non compressed files (with extension `.vol`), set `comp = NPIC_COMPRESS_NONE`. You can directly read or write compressed files, having extension `.vol.gz`, `.vol.bz2` or `.vol.7z`, by setting `comp = NPIC_COMPRESS_GZ`, `NPIC_COMPRESS_BZ2` or `NPIC_COMPRESS_7Z`.

• **NpicWriteVOL**

```
int NpicWriteVOL (Npic_image *np, const char *filename, Npic_file_compress comp);
```

Write file in VOL format and save Npic_image image of type `NPIC_IMAGE_3C` or `3L`.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

• **NpicReadVOL**

```
Npic_image *NpicReadVOL (const char *filename, Npic_file_compress comp);
```

Read file in VOL format and create Npic_image image of type `NPIC_IMAGE_3C` or `3L`.

Return image on success, else return `NULL`. Verbose.

• **NpicInfoVOL**

```
int NpicInfoVOL (const char *filename, Npic_file_compress comp);
```

Print infos of a file in VOL format.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

1.3.5 PNM format for 2D images (PBM/PGM/PPM)

Sources: files_pnm.h , files_pnm.c , files_hints.h , files_hints.c .

The PNM file formats stand for "Portable aNyMap file format" and have `.pnm` extension in the general case, or `.pbm`, `.pgm` and `.ppm` extensions, respectively, for bitmap 1 bit format, greylevel 8 or 16 bit format, pixmap 8+8+8 or 16+16+16 bit format. See `man pnm/pgm/pbm/ppm` for specifications.

For non compressed files (with extension `.pnm` or others), set `comp = NPIC_COMPRESS_NONE`. You can directly read or write compressed files, having extension `.pnm.gz`, `.pnm.bz2` or `.pnm.7z`, by setting `comp = NPIC_COMPRESS_GZ`, `NPIC_COMPRESS_BZ2` or `NPIC_COMPRESS_7Z`.

• **Hints**

This file format can be managed with *hints* at the end of the file name. A hint is a list of comma-separated words after a ":" and without blanks. The hints are memorized and suppressed from the name. For instance in a file named `"tmp1.pnm.gz:16-bit,little-endian"`, the real name is `"tmp1.pnm.gz"` and the hints are `"16-bit"` and `"little-endian"`.

To get the available hints list, try writing or saving with the hint `help`. This prints a help message and stops with an error.

The allowed hints when writing a PNM file with `NpicWriteImage` are:

- ▷ `bitmap` (default is greymap or pixmap);
- ▷ `ascii` (default is binary);
- ▷ `16-bit` (default is 8-bit);
- ▷ `little-endian` (for 16-bit pixels; default is big-endian).

The allowed hints when reading a PNM file with `NpicReadImage` are:

▷ `little-endian` (for 16-bit pixels; default is big-endian).

Hints should only be used with `NpicReadImage` and `NpicWriteImage`, but *not* with the functions below.

• **NpicWritePNM**

```
int NpicWritePNM (Npic_image *np, const char *filename, Npic_file_compress comp,
    Npic_pnm_format format, Npic_pnm_option option)
```

Save a 2C, 2L or 2Q image `np` in a PNM file.

`format` is one of `NPIC_PNM_BITMAP_ASCII`, `NPIC_PNM_GREYMAP_ASCII`, `NPIC_PNM_PIXMAP_ASCII`, `NPIC_PNM_BITMAP_BINARY`, `NPIC_PNM_GREYMAP_BINARY`, `NPIC_PNM_PIXMAP_BINARY`, `NPIC_PNM_AUTO_DETECT`. In the latter case, the format is guessed from the options.

`option` is either `NPIC_PNM_OPTION_NONE`, either a bitwise union of `NPIC_PNM_BITMAP` (default is greymap), `NPIC_PNM_ASCII` (default is binary), `NPIC_PNM_ALLOW_16_BIT` (default is 8 bit), `NPIC_PNM_LITTLE_ENDIAN` (default is big endian).

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

• **NpicReadPNM**

```
Npic_image *NpicReadPNM (const char *filename, Npic_file_compress comp,
    Npic_pnm_option option)
```

Read a PNM file and create a 2C, 2L or 2Q image.

`option` is `NPIC_PNM_OPTION_NONE`, or `NPIC_PNM_LITTLE_ENDIAN` (default is big endian).

Return image on success, else return `NULL`. Verbose.

• **NpicInfoPNM**

```
int NpicInfoPNM (const char *filename, Npic_file_compress comp);
```

Print infos about a PNM file.

Return `NPIC_SUCCESS` on success, else return error code. Verbose.

As an example, see test-pnm.c

1.3.6 Printing an image in ascii

Sources: files_print.h , files_print.c .

Work only on 2D images.

• **NpicPrintImage***

```
int NpicPrintImage (Npic_image *np);
int NpicPrintImageB(Npic_image *np);
```

Print whole image to stdout, without or with border pixels.

Return NPIC_SUCCESS on success, else return error code. Verbose.

• **NpicPrintImage*F**

```
int NpicPrintImageF (Npic_image *np, FILE *f);
int NpicPrintImageBF (Npic_image *np, FILE *f);
```

Print whole image to a file f, without or with border pixels. The file must be opened before calling, and closed after.

Return NPIC_SUCCESS on success, else return error code. Verbose.

• **NpicPrintImageF_YX**

```
int NpicPrintImageF_YX (Npic_image *np, FILE *f,
                        int y1, int y2, int x1, int x2);
```

The same for a bounding box y1 <= y <= y2 and x1 <= x <= x2.

Return NPIC_SUCCESS on success, else return error code. Verbose.

As an example, see test-mirror.c.

1.4 Masks

1.4.1 Creation of a mask

Sources: mask_creat.h , mask_creat.c.

• **NpicCreateMask**

```
Npic_mask *NpicCreateMask (int ntype);
```

Creation of a mask, with current number of vectors set to 0.

Return the mask on success, else return NULL. Verbose.

The allowed types are NPIC_MASK_2L, NPIC_MASK_2D, NPIC_MASK_3L, NPIC_MASK_3D, NPIC_MASK_4L, NPIC_MASK_4D, NPIC_MASK_5L, NPIC_MASK_5D, NPIC_MASK_6L, NPIC_MASK_6D.

Usage:

```
Npic_mask *nm1;
nm1 = NpicCreateMask (NPIC_MASK_..);
if (nm1 == NULL) ...
....
NpicDestroyMask (nm1);
```

• **NpicCreateMaskDP**

```
Npic_mask *NpicCreateMaskDP (int dim, int pixeltype);
```

Creation of a mask, having dimension `dim` (in 2..6), and `pixeltype` being `NPIC_C` or `NPIC_L`. The current number of vectors is set to 0.

Return the mask on success, else return `NULL`. Verbose.

Usage:

```
Npic_mask *nm1;
nm1 = NpicCreateMaskDP (2, NPIC_L);
if (nm1 == NULL) ...
....
NpicDestroyMask (nm1);
```

- **NpicDestroyMask**

```
void NpicDestroyMask (Npic_mask *nm1);
```

Destruction of a mask (even if it is not ok). Silent.

- **NpicCopyMask**

```
Npic_mask *NpicCopyMask (Npic_mask *src);
```

Copy a mask `src` to a mask `dest`. Do nothing if `src` is not ok. Do not copy text properties; for that, use `NpicMaskCopyProps`.

Return the mask on success, else return `NULL`. Verbose.

1.4.2 Checks on a mask

Sources: mask_check.h , mask_check.c.

- **NpicMaskIsOK**

```
int NpicMaskIsOK (Npic_mask *m, const char *funcname);
```

Check if mask is not `NULL`, has correct type and is ok.

Return `NPIC_SUCCESS` on success, else print a message containing `funcname` and return error code.

- **NpicMaskCompat**

```
int NpicMaskCompat (Npic_mask *m, Npic_mask *np);
```

Check if mask type and mask type are compatible (even if not ok).

Return `NPIC_SUCCESS`, else error code. Silent.

1.4.3 Type to name and vice-versa

Sources: mask_ntype.h , mask_ntype.c .

• **NpicMaskNtype**

```
int NpicMaskNtype (int dim, int pixeltype);
```

Describe the mask type corresponding to `dim` and `pixeltype`, where `dim` is in 2..6 and `pixeltype` is one of `NPIC_L`, `NPIC_D`.

Return one of these constants: `NPIC_MASK_2L`, .. `NPIC_MASK_6D`, else `NPIC_NONE` on error. Silent.

• **NpicMaskPixelType**

```
int NpicMaskPixelType (int ntype)
```

Describe the pixel type from the mask type `ntype`, where `ntype` is one of `NPIC_MASK_2D` .. `NPIC_MASK_6D`.
Return one of these constants: `NPIC_L`, `NPIC_D`, else `NPIC_NONE` on error. Silent.

• **NpicMaskTypeName**

```
const char *NpicMaskTypeName (int ntype);
```

Return a static string containing the name of the mask type, such as `"NPIC_MASK_2L"`, else `"ERROR"`. `ntype` is one of `NPIC_MASK_2L` .. `NPIC_MASK_6D`. Silent.

• **NpicMaskDim**

```
int NpicMaskDim (int type);
```

Return the dimension from the type, or -1. Silent.

1.4.4 Inserting vectors

Sources: mask_creat.h , mask_creat.c.

• **NpicMaskInsert_***

```
void NpicMaskInsert_2l (Npic_mask *m,        int y, int x, Npic_l h);
void NpicMaskInsert_2d (Npic_mask *m,        int y, int x, Npic_d h);
void NpicMaskInsert_3l (Npic_mask *m, int z, int y, int x, Npic_l h);
void NpicMaskInsert_3d (Npic_mask *m, int z, int y, int x, Npic_d h);
void NpicMaskInsert_4l (Npic_mask *m,
                              int t, int z, int y, int x, Npic_l h);
void NpicMaskInsert_4d (Npic_mask *m,
                              int t, int z, int y, int x, Npic_d h);
void NpicMaskInsert_5l (Npic_mask *m,
```

```
                          int s, int t, int z, int y, int x, Npic_l h);
void NpicMaskInsert_5d (Npic_mask *m,
                          int s, int t, int z, int y, int x, Npic_d h);
void NpicMaskInsert_6l (Npic_mask *m,
                 int r, int s, int t, int z, int y, int x, Npic_l h);
void NpicMaskInsert_6d (Npic_mask *m,
                 int r, int s, int t, int z, int y, int x, Npic_d h);
```

Insert a vector in the mask `m`, and expand its internal list if necessary. `h` is a weight associated to the vector.

Do nothing if `m` is not ok. set not ok on error. Verbose.

1.4.5 Computing masks

Sources: mask_comp.h , mask_comp.c.

• **NpicCompHalfMask**

```
Npic_mask *NpicCompHalfMask(Npic_mask *mg);
```

Create and compute the half mask corresponding to the mask generator `mg`, by inserting all the G-symmetrical vectors of `mg` which are in the half space in scan line order `"for t, for z, for y, for x"`.

Do nothing if `mg` is not ok. Do not copy text properties; for that, use `NpicMaskCopyProps`.

Return the new mask on success, else `NULL`. Verbose.

• **NpicCompFullMask**

```
Npic_mask *NpicCompFullMask(Npic_mask *mg);
```

Create and compute the full mask corresponding to the mask generator `mg`, by inserting all the G-symmetrical vectors of `mg`.

Do nothing if `mg` is not ok. Do not copy text properties; for that, use `NpicMaskCopyProps`.

Return the new mask on success, else `NULL`. Verbose.

• **NpicCompGeneMask**

```
Npic_mask *NpicCompGeneMask(Npic_mask *m);
```

Create and compute the mask generator corresponding to the half or full mask `m`. Assume m is G-symmetric.

Do nothing if m is not ok. Do not copy text properties; for that, use `NpicMaskCopyProps`.

Return the new mask on success, else NULL. Verbose.

• **NpicMaskLargestCoord**

```
int NpicMaskLargestCoord (Npic_mask *m);
```

Compute the largest coordinate `r` of the vectors in the mask (even if `m` is not ok).

Return `r` on success, else 0. Silent.

1.4.6 Accessing to vectors

Sources: mask_creat.h , mask_creat.c.

• **NpicCastMask**

```
void *NpicCastMask (Npic_mask *m);
```

Return the mask which is inside the `Npic_mask` structure (even if not ok). Silent.

We call *n-mask* a mask of type `Npic_mask`, which can store a mask having any of the allowed dimensions and pixel types. A *typed mask* is a variable of type `Npic_mask_`*dt* having a determined dimension *d* and pixel type *t*.

The vectors cannot be accessed in a n-mask. Since `Npic_mask` is a union of structs, we normally should write `Npic_mask_`*dt*`*m1 = &nm1->dm_`*dt* to get the typed mask `m1` from `nm1`; thanks to this function, we can simplier write `Npic_mask_`*dt*`*m1 = NpicCastMask (nm1)`. How does it work ? This is actually a simple cast of the `Npic_mask*` type to `void*`.

Of course, `NpicCastMask` must be called *after* successful creation of a mask:

```
Npic_mask   *nm1;   /* the n-mask     */
Npic_mask_2l *m1;   /* the typed mask */

nm1 = NpicCreateMask (NPIC_MASK_2L);
if (nm1 == NULL) return;

NpicMaskInsert_2l (nm1, 0, 1, 5);
NpicMaskInsert_2l (nm1, 1, 1, 7);

m1 = NpicCastMask (nm1);
for (i = 0; i < m1->nb; i++)
    printf ("y = %d  x = %d  h = %"NPIC_LP"\n",
        m1->v[i].y, m1->v[i].x, m1->v[i].h);

NpicDestroyMask (nm1);
```

To printf a weight of type `Npic_d`, use format `%lf` ; when the type is `Npic_l`, use format `%"NPIC_LP"` (`NPIC_LP` is a macro, its value is `d` or `ld`, depending on the actual type of `Npic_l`: int or long).

1.4.7 Printing

Sources: mask_creat.h , mask_creat.c.

• **NpicPrintMask**

```
void NpicPrintMask (Npic_mask *m);
```

Print a mask to stdout. Do not print if not ok. Verbose.

As an example, see test-mask.c

1.4.8 NMASK format for masks

Sources: files_nmask.h , files_nmask.c .

The NMASK file format (extension `.nmask`) is a proposed file format for storing masks, such as distance masks, medial axis test neighbourhood, convolution masks, etc, whatever the dimension and the pixel type.

The beginning of the file is a text header describing the content and the text properties, the next is a text list of vectors and weights. See section 3.2 for specifications.

The properties allow to store additionnal datas, such as the purpose of the mask, or if symmetries must be computed before use, etc. Properties are not limited in number neither in size, see section 1.7.3

• **NpicWriteMask**

```
int NpicWriteMask (Npic_mask *m, const char *filename);
```

Save `Npic_mask mask` in a file with extension ".nmask" of ".nmask.gz" (compressed on fly).

Return `NPIC_SUCCESS` on success, return error code. Verbose.

• **NpicReadMask**

```
Npic_mask *NpicReadMask (const char *filename);
```

Read a file with extension ".nmask" of ".nmask.gz" (uncompressed on fly) and create a mask.

Return mask on success, else `NULL`. Verbose.

• **NpicInfoMask**

```
int NpicInfoMask (const char *filename);
```

Print infos of a file in ".nmask" of ".nmask.gz" format.

Return `NPIC_SUCCESS` on success, else error code. Verbose.

• **NpicReadDistanceMask**

```
Npic_mask *NpicReadDistanceMask (const char *filename);
```

Read a file in ".nmask" of ".nmask.gz" format, check in its properties if it is a distance mask, create appropriate mask and add G-symmetries if needed.

Return mask on success, else `NULL`. Verbose.

• **NpicReadMedialAxisMask**

```
Npic_mask *NpicReadMedialAxisMask (const char *filename);
```

Read a file in ".nmask" of ".nmask.gz" format, check in its properties if it is a medial axis mask (a.k.a MLut or neighbourhood test), create appropriate mask.

The "RVerified" text property stores the largest verified radius. If absent, or less than the largest radius, the latter is stored.

Return mask on success, else `NULL`. Verbose.

To create and manage masks, see npic-mask.c .

1.4.9 G-symmetries

For sources, see mask_gsym.h , mask_gsym.c .

• **NpicGsymPermu Init / Next**

```
void NpicGsymPermuInit (Npic_gsym *gs);
int NpicGsymPermuNext (Npic_gsym *gs);
```

Johnson's algorithm to compute all permutations in `[1 .. dim]` see (Johnson 1963) and (R. Seroul 1995). This function also check if `(x[1], .., x[dim])` is unique.

• **NpicGsymCombi Init / Next**

```
void NpicGsymCombiInit (Npic_gsym *gs);
int NpicGsymCombiNext (Npic_gsym *gs);
```

Compute all the distinct sign combinations of `(x[1], .., x[dim])` which are in the half-mask in scan line order (when `half==1`), or in the full-mask (when `half==0`).

The scan line order is : `for t[dim], for t[dim-1], .., for t[1].` A point `(x[1], .., x[j], 0 ... 0)` belongs to the half mask iff `x[j]` > 0.

• **NpicGsym Init / Next**

```
void NpicGsymInit (Npic_gsym *gs);
int NpicGsymNext (Npic_gsym *gs);
```

Compute all the distinct G-symmetries of `(x[1], .., x[dim])` in the half (or full) mask. The G-symmetries are all the permutations and sign combinations.

• **NpicGsymPrint**

```
void NpicGsymPrint (Npic_gsym *gs);
```

Print positions `pos[]`, directions `dir[]` and coordinates `x[]`.

Usage:

```
Npic_gsym gs;
gs.x[1] = 1; gs.x[2] = 3; gs.x[3] = 0; gs.dim = 3; gs.half = 1;

NpicGsymInit (&gs);
while (NpicGsymNext (&gs))
    NpicGsymPrint (&gs);
```

As an example, see test-gsym.c

1.5 Computations on images

1.5.1 Comparison of pixels

Sources: calc_diff.h , calc_diff.c .

• **NpicCompare**

```
int NpicCompare (Npic_image *np1, Npic_image *np2, int n);
```

Compare two images, pixel per pixel (border pixels excluded).

Return <0 if images are not ok, or do not have same type and size, 0 if same pixel values, >0 the number of pixels which differ.

Print at most `n` differences. Verbose.

• **NpicCompareBin**

```
int NpicCompareBin (Npic_image *np1, Npic_image *np2, int n);
```

The same, but compare pixel binary values (== 0 or != 0).

1.5.2 Changing values

Sources: calc_values.h , calc_values.c .

The following functions do not modify border pixels; they do nothing if `np` is not ok, set not ok on error, and are verbose.

Depending on image type, use *function*`_i` for integer pixels, *function*`_d` for double pixels, *function*`_q` for 4 short pixels.

• **NpicFillVal_***

```
void NpicFillVal_i (Npic_image *np, int val1);
void NpicFillVal_d (Npic_image *np, double val1);
void NpicFillVal_q (Npic_image *np, int a, int b, int c, int d);
```

Set all pixels to the value `val1`.

• **NpicChangeVal_***

```
void NpicChangeVal_i (Npic_image *np, int val1, int val2);
void NpicChangeVal_d (Npic_image *np, double val1, double val2);
void NpicChangeVal_q (Npic_image *np, int a1, int b1, int c1, int d1,
                                      int a2, int b2, int c2, int d2);
```

Set each pixel having value `val1` to the value `val2`.

• **NpicExchangeVal_***

```
void NpicExchangeVal_i (Npic_image *np, int val1, int val2);
void NpicExchangeVal_d (Npic_image *np, double val1, double val2);
void NpicExchangeVal_q (Npic_image *np, int a1, int b1, int c1, int d1,
                                        int a2, int b2, int c2, int d2);
```

Set each pixel having value `val1` to the value `val2`, and vice versa.

• **NpicMin/MaxValue**

```
double NpicMaxValue (Npic_image *np1);
double NpicMinValue (Npic_image *np1);
```

Return the max or min of pixels values in image interior.

1.5.3 Thresholding pixel values

Sources: calc_threshold.h , calc_threshold.c .

The following functions do not modify border pixels; they do nothing if `np` is not ok, set not ok on error, and are verbose.

Depending on image type, use *function*`_i` for integer pixels, *function*`_d` for double pixels, *function*`_q` for 4 short pixels.

• **NpicThresholdLT_***

```
void NpicThresholdLT_i (Npic_image *np, int val1, int val2);
void NpicThresholdLT_d (Npic_image *np, double val1, double val2);
void NpicThresholdLT_q (Npic_image *np, int a1, int b1, int c1, int d1,
                                        int a2, int b2, int c2, int d2);
```

Set each pixel having value $<$ `val1` to the value `val2`.

• **NpicThresholdLE_***

```
void NpicThresholdLE_i (Npic_image *np, int val1, int val2);
void NpicThresholdLE_d (Npic_image *np, double val1, double val2);
void NpicThresholdLE_q (Npic_image *np, int a1, int b1, int c1, int d1,
                                        int a2, int b2, int c2, int d2);
```

Set each pixel having value <= `val1` to the value `val2`.

• **NpicThresholdGT_***

```
void NpicThresholdGT_i (Npic_image *np, int val1, int val2);
void NpicThresholdGT_d (Npic_image *np, double val1, double val2);
void NpicThresholdGT_q (Npic_image *np, int a1, int b1, int c1, int d1,
                                        int a2, int b2, int c2, int d2);
```

Set each pixel having value > `val1` to the value `val2`.

• **NpicThresholdGE_***

```
void NpicThresholdGE_i (Npic_image *np, int val1, int val2);
void NpicThresholdGE_d (Npic_image *np, double val1, double val2);
void NpicThresholdGE_q (Npic_image *np, int a1, int b1, int c1, int d1,
                                        int a2, int b2, int c2, int d2);
```

Set each pixel having value >= `val1` to the value `val2`.

1.5.4 Summing images

Sources: calc_algeb.h , calc_algeb.c .

The following functions do not modify border pixels. The images must have same type and size; they can have same address in memory.

The functions do nothing if one of the image is not ok; they set not ok for `dest` on error, and are verbose.

• **NpicLinearSum**

```
void NpicLinearSum (Npic_image *dest, Npic_image *src1, double c1, double c2);
```

Set each pixel of `dest` to a linear combination of corresponding pixels in `src1` :
`dest := src1 * c1 + c2` .

• **NpicBilinearSum**

```
void NpicBilinearSum (Npic_image *dest, Npic_image *src1, Npic_image *src2,
                double c1, double c2, double c3);
```

Set each pixel of `dest` to a linear combination of corresponding pixels in `src1` and `src2` :
`dest := src1 * c1 + src2 * c2 + c3` .

• **NpicTrilinearSum**

```
void NpicTrilinearSum (Npic_image *dest, Npic_image *src1, Npic_image *src2,
                Npic_image *src3, double c1, double c2, double c3, double c4);
```

Set each pixel of `dest` to a linear combination of corresponding pixels in `src1`, `src2` and `src3` :
`dest := src1 * c1 + src2 * c2 + src3 * c3 + c4` .

• **NpicQuadraticSum**

```
void NpicQuadraticSum (Npic_image *dest, Npic_image *src1,
                       double c1, double c2, double c3);
```

Set each pixel of `dest` to a quadratic combination of corresponding pixels in `src1` :
`dest := src1^2 * c1 + src1 * c2 + c3` .

• **NpicBiQuadraticSum**

```
void NpicBiQuadraticSum (Npic_image *dest, Npic_image *src1, Npic_image *src2,
                 double c1, double c2, double c3, double c4, double c5);
```

Set each pixel of `dest` to a quadratic combination of corresponding pixels in `src1` and `src2` :
`dest := src1^2 * c1 + src1 * c2 + src2^2 * c3 + src2 * c4 + c5` .

1.5.5 Drawing into images

Sources: calc_draw.h , calc_draw.c .

The following functions do not modify border pixels; they do nothing if `np` is not ok, set not ok on error, and are verbose.

Depending on image dimension and pixels type, use appropriate function.

• **NpicDrawRect_***

```
void NpicDrawRect_?c (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_c val1);

void NpicDrawRect_?l (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_l val1);

void NpicDrawRect_?d (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_d val1);

void NpicDrawRect_?q (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    int a1, int b1, int c1, int d1);
```

Draw a rectangle bounded by P and Q, with pixel value `val1`. P and Q can be outside image. Replace `?` by dimension: 2, 3, 4, 5 or 6.

• **NpicDrawEllipse_***

```
void NpicDrawEllipse_?c (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_c val1);

void NpicDrawEllipse_?l (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_l val1);

void NpicDrawEllipse_?d (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_d val1);

void NpicDrawEllipse_?q (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    int a1, int b1, int c1, int d1);
```

Draw an ellipse bounded by P and Q, with pixel value `val1`. P and Q can be outside image. Replace `?` by dimension: 2, 3, 4, 5 or 6.

1.5.6 Drawing lines

Sources: calc_bresenham.h , calc_bresenham.c .

The following functions do not modify border pixels; they do nothing if `np` is not ok, set not ok on error, and are verbose.

Depending on image dimension and pixels type, use appropriate function.

• **NpicDrawLine_***

```
void NpicDrawLine_?c (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_c cP, Npic_c cQ);

void NpicDrawLine_?l (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_l cP, Npic_l cQ);

void NpicDrawLine_?d (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
    Npic_d cP, Npic_d cQ);

void NpicDrawLine_?q (Npic_image *np,
    ..., int yP, int xP, ..., int yQ, int xQ,
```

```
int aP, int bP, int cP, int dP, int aQ, int bQ, int cQ, int dQ);
```

Draw a discrete segment from P and Q, with pixel values from `cP` to `cQ`, using generalized Bresenham's algorithm. P and Q must be inside image. Replace `?` by dimension: 2, 3, 4, 5 or 6.

1.6 Distance Transformations

1.6.1 Front-end for Distance Transformations (DT)

Sources: dist_dt.h , dist_dt.c .

A distance transform (DT) is a copy of a binary image where each foreground point (non-zero points) is labelled to its distance to the background (zero points).

This section presents a front-end for distance transformations, allowing to compute DTs for weighted distances or Euclidean distances. The distance is defined by a mask `mh`, which is either a half weighted distance mask (the property `DistanceType` is `Weighted`), or a Euclidean distance mask, that is an empty mask with property `DistanceType` set to `SquaredEuclidean`, etc. See section 3.2 for available distance types.

A number of distance masks is available in masks/. The weighted distance masks are starting with "d-", followed by integer weights; the Euclidean distance masks are starting with "d-sed" or "d-ced" or "d-fed" (Squared, Ceil, Floor Euclidean Distance, resp.).

The following functions do nothing if `np` is not ok. They set not ok on error and are verbose. They return computation time in *second.microsecond*. Available for images 2L, 3L, 4L, 5L, 6L and 2D, 3D, 4D, 5D, 6D.

• **NpicDT**

```
double NpicDT (Npic_image *np, Npic_mask *mh);
```

Compute the distance transform (DT) on image `np`.

• **NpicDT_inf**

```
double NpicDT_inf (Npic_image *np, Npic_mask *mh);
```

Compute the distance transform (DT_inf) on image `np`, with "infinite border", i.e border pixels propagating no distance value.

• **NpicDT_rev**

```
double NpicDT_rev (Npic_image *np, Npic_mask *mh);
```

Compute the reverse distance transform (DT_rev) on image `np`.

The functions which actually perform the computations are presented bellow.

1.6.2 Squared Euclidean Distance Transformation (SEDT)

Sources: dist_sedt_hirata.h , dist_sedt_hirata.c , dist_sedt_saito.h , dist_sedt_saito.c , dist_sedt_quadra.h , dist_sedt_quadra.c .

The following functions do nothing if `np` is not ok. They set not ok on error and are verbose. They return computation time in *second.microsecond* . Available for images 2L, 3L, 4L, 5L and 6L.

All these function are multi-threaded with OpenMP (if enabled, see README).

• **NpicSEDT_***

```
double NpicSEDT_Hirata (Npic_image *np);
double NpicSEDT_Saito  (Npic_image *np);
double NpicSEDT_quadra (Npic_image *np);
```

Compute the Squared Euclidean Distance Transform (SEDT) on image `np` :

- Use the optimal in time linear algorithm from Hirata [2] (and some fixes);
- use the separable algorithm from Saito [1];
- quadratic algorithm.

• **NpicSEDT_*_inf**

```
double NpicSEDT_Hirata_inf (Npic_image *np);
double NpicSEDT_Saito_inf  (Npic_image *np);
double NpicSEDT_quadra_inf (Npic_image *np);
```

Do the same, but with "infinite border", i.e border pixels propagating no distance value:

- Derived from the optimal in time linear algorithm from Hirata [2];
- derived from the separable algorithm from Saito [1];
- quadratic algorithm.

• **NpicSEDT_*_rev**

```
double NpicSEDT_Hirata_rev (Npic_image *np);
double NpicSEDT_Saito_rev  (Npic_image *np);
double NpicSEDT_quadra_rev (Npic_image *np);
```

Compute the Reverse Squared Euclidean Distance Transform (RevSEDT) on image `np`:

- Derived from the optimal in time linear algorithm from Hirata [2] (see also [3]);
- derived from the separable algorithm from Saito [1];
- quadratic algorithm.

As an example, see npic-sedt.c

The Hirata's [2] and Saito's *et al.* [1] algorithms belongs to the family of separable transforms: they operate independently in each dimension. [1] was the first n-dimensional, exact and efficient SEDT, but still a non-linear algorithm. [2] is the linearized version of [1], and [3] is an extension of [2].

For SEDT and SEDT_inf, [1] is a little faster than [2] for small images, while [2] becomes much faster than [1] for large images. For SEDT_rev, [2] is generally much faster than [1] for all sizes.
The quadratic algorithms are very slow, but given for checks.

References:

1 T. Saito et J.I. Toriwaki, 1994. "New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications". *Pattern Recognition*, 27(11):1551-1565.

2 T. Hirata, 1996. "A unified linear-time algorithm for computing distance maps". *Information Processing Letters*, 58:129-133.

3 D. Coeurjolly, 2003. "d-Dimensional Reverse Euclidean Distance Transformation and Euclidean Medial Axis Extraction in Optimal Time". In *11th DGCI*, LNCS vol. 2886, Springer-Verlag, p. 327-337, Naples, Italy.

1.6.3 Weighted Distance Transformation (WDT)

Sources: dist_wdt.h , dist_wdt.c .

The following functions do nothing if `np` is not ok. They set not ok on error and are verbose. They return computation time in *second.microsecond* . Available for images 2L, 3L, 4L, 5L, 6L and 2D, 3D, 4D, 5D, 6D.

• **NpicWDT**

```
double NpicWDT (Npic_image *np, Npic_mask *mh);
```

Compute the weighted distance transform (WDT) on image `np` for the half chamfer mask `mh`.

• **NpicWDT_inf**

```
double NpicWDT_inf (Npic_image *np, Npic_mask *mh);
```

Compute the weighted distance transform (WDT_inf) on image `np` for the half chamfer mask `mh`, with "infinite border", i.e border pixels propagating no distance value.

• **NpicWDT_rev**

```
double NpicWDT_rev (Npic_image *np, Npic_mask *mh);
```

Compute the reverse weighted distance transform (WDT_rev) on image `np` for the half chamfer mask `mh`.

As an example, see npic-wdt.c

References:

1 A. Rosenfeld and J.L. Pfaltz, 1966. Sequential operations in digital picture processing. *Journal of ACM*, 13(4):471-494.

2 U. Montanari, 1968. A method for obtaining skeletons using a quasi-euclidean distance. *Journal of ACM*, 15:600-624.

3 G. Borgefors, 1984. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics and Image Processing*, 27:321-345.

1.6.4 Medial Axis Extraction

Sources: mlut_ma.h , mlut_ma.c .

The Medial Axis (MA) of shape S is the set of maximal balls in S, a ball B being maximal in S if B is included in S, and if B is not included in any other ball included in S. As MA is a covering of S, MA is a reversible transform.

The MA can be efficiently extracted from a DT, using a test neighbourhood (denoted Mlut) and a look-up-table (denoted Lut). The easiest way to compute MA is to use the first function bellow (`NpicExtractMA_bin`).

The following functions are available for images 2L, 3L, 4L, 5L and 6L.

• **NpicExtractMA_bin**

```
void NpicExtractMA_bin (Npic_image *np1, Npic_image *nMA,
                        Npic_mask *nMdist, Npic_mask *nMlut)
```

Compute a DT for a binary object, then extract Medial Axis from this DT.

Input: `np1` a binary image, `nMdist` a distance mask (half weighted or Euclidean), `nMlut` a MA test neighboorhood or NULL.
Output: `np1` contains the DT, `nMA` contains the medial axis, `nMlut` (if it wasn't NULL) is computed or updated.

Do nothing if images or masks are not ok. set not ok for MA on error. Verbose.

As an example, see npic-dtma.c.

• **NpicExtractMA_DT**

```
void NpicExtractMA_DT (Npic_image *nDT, Npic_image *nMA,
                       Npic_mask *nMdist, Npic_mask *nMlut)
```

Extract the Medial Axis from a DT.

Input: `nDT` a DT, `nMdist` a distance mask (half weighted or Euclidean), `nMlut` a MA test neighboorhood or NULL.
Output: `nMA` contains the medial axis, `nMlut` (if it wasn't NULL) is computed or updated.

Do nothing if images or masks are not ok. set not ok for MA on error. Verbose.

• **NpicExtractMA_Lut**

```
void NpicExtractMA_Lut (Npic_image *nDT, Npic_image *nMA,
                        Npic_mask *nMlut, Npic_Lut *Lut, Npic_l Rmax);
```

Extract Medial Axis from a DT having Lut completely pre-computed.

Do nothing if `nDT`, `nMA` or `nMlut` is not ok. set not ok for MA on error. Verbose.

• **NpicExtractMA_Mlut**

```
void NpicExtractMA_Mlut (Npic_image *nDT, Npic_image *nMA,
                         Npic_mask *nMlut, Npic_image *nCTg, Npic_l Rmax);
```

Extract Medial Axis from a DT having Mlut pre-computed, compute and store one Lut column at a time.

Do nothing if `nDT`, `nMA` or `nMlut` is not ok. set not ok for MA on error. Verbose.

References:

1 G. Borgefors, I. Ragnemalm, and G. Sanniti di Baja, 1991. "The Euclidean Distance Transform : finding the local maxima and reconstructing the shape". In 7th Scandinavian Conf. on Image Analysis, volume 2, pages 974-981, Aalborg, Denmark.

2 G. Borgefors, 1993. "Centres of maximal disks in the 5-7-11 distance transform". In 8th Scandinavian Conf. on Image Analysis, pages 105-111, Tromso, Norway.

3 E. Remy and E. Thiel, 2005. Exact Medial Axis with Euclidean Distance. Image and Vision Computing, 23(2):167-175. Link.

4 E. Remy and E. Thiel, 2002. Medial axis for chamfer distances: computing look-up tables and neighbourhoods in 2D or 3D. Pattern Recognition Letters, 23(6):649-661. Link.

1.6.5 Medial Axis - LUT and Mlut

Sources: mlut_remy.h , mlut_remy.c .

Additionnal sources: mlut_sedtg.h , mlut_sedtg.c , mlut_wdtg.h , mlut_wdtg.c.

These functions show the detail of the Medial Axis, Mlut and Lut computations.

The following functions are available for images 2L, 3L, 4L, 5L and 6L.

• **NpicMlutCompLutCol**

```
double NpicMlutCompLutCol (Npic_image *nCTg, Npic_vec *vg,
                           Npic_l Rmax, Npic_l *LutCol);
```

Lut column Computation Algorithm. See [3, chap.5 p.79], [1], [2].

Input: `nCTg` the distance cone to origin, `vg` the vector, `Rmax` the greatest radius to be verified in Lut.
Output: The Lut column `LutCol` is filled with the correct values.

Assume that `nCTg` is a `Lñ` image and that `LutCol` is `[0..Rmax]`.
Do nothing if `nCTg` is not ok. set not ok on error. Verbose.

Return computation time in second.microsecond

• **NpicMlutInitLut**

```
int NpicMlutInitLut (Npic_Lut *Lut, Npic_l Rtarget);
```

Initialize the `Lut` structure with no column. The future columns will have `[0..Rtarget]` bounds.

Return `NPIC_SUCCESS` or error code. Verbose.

• **NpicMlutNewCol**

```
int NpicMlutNewCol (Npic_Lut *Lut);
```

Create and Insert a column in `Lut`.

Return `NPIC_SUCCESS` or error code. Verbose.

• **NpicMlutFreeLut**

```
void NpicMlutFreeLut (Npic_Lut *Lut);
```

Free `Lut` and its columns. Verbose.

• **NpicMlutCompLutMask**

```
int NpicMlutCompLutMask (Npic_image *nCTg, Npic_mask *nMlut, Npic_Lut *Lut,
                         Npic_l Rknown, Npic_l Rtarget, Npic_mask *nMg);
```

Full Computation Algorithm of Lut and Mlut for SEDT [1] or WDT [2].

Input: `nCTg` a `Lñ` image, `nMlut` the generator of the Lut mask, `Lut` the look-up table, `Rknown` the last verified radius, `Rtarget` the maximum radius to be verified, `nMg == NULL` for SEDT, else the generator of the WD mask.
Output: `nMlut` and `Lut` are filled with the correct values.

Do nothing if `nCTg` is not ok. Verbose. Return error code.

Usage: To compute `nMlut` and `Lut` from beginning to `Rtarget` with `L` such that `L*L` > `Rtarget` for SED :

```
Npic_mask  *nMlut = NpicCreateMask (NPIC_MASK_..);
Npic_image *nCTg  = NpicCreateImage_.. (L, L, ..., 0, 0, ...);
Npic_Lut Lut;
char s[200];

NpicMaskAddProp (nMlut, "Nature", "MedialAxis");
NpicMaskAddProp (nMlut, "RVerified", "0");
NpicMlutCTg_SED (nCTg);
NpicMlutInitLut (&Lut, Rtarget);

NpicMlutCompLutMask (nCTg, nMlut, &Lut, 0, Rtarget, NULL);

sprintf (s, "%d", Rtarget);
```

```
NpicMaskAddProp (nMlut, "RVerified", s);
NpicWriteMask (nMlut, "name.nmask");

NpicDestroyMask (nMlut);
NpicDestroyImage (nCTg);
NpicMlutFreeLut (&Lut);
```

• **NpicAllocAndCompLutCol**

```
Npic_l *NpicAllocAndCompLutCol (Npic_mask *nMlut, Npic_image *nCTg,
                                Npic_l Rmax, int n);
```

Create and compute one Lut column. Return the column or NULL. The result must be freed after use.

Do nothing if `nDT`, `nCTg` or `nMlut` is not ok. Verbose.

References:

1 E. Remy and E. Thiel, 2005. Exact Medial Axis with Euclidean Distance. Image and Vision Computing, 23(2):167-175. Link.

2 E. Remy and E. Thiel, 2002. Medial axis for chamfer distances: computing look-up tables and neighbourhoods in 2D or 3D. Pattern Recognition Letters, 23(6):649-661. Link.

3 E. Thiel, 1994. Les distances de chanfrein en analyse d'images : fondements et applications. These de Doctorat, Universite Joseph Fourier, Grenoble 1. Link.

1.7 Miscellaneous

1.7.1 Error messages

Sources: errors.h , errors.c .

• **NpicStrErr**

```
const char *NpicStrErr (int err);
```

Return the string message corresponding to error-code `err`.

• **NpicError**

```
int NpicError (const char *funcname, int err, const char *format, ...);
```

Return `err` and, `if err != NPIC_SUCCESS`, print error message on stderr:

▷ the string "libNpic", then

▷ the name `funcname` of the function if non empty, then

- ▷ the string message corresponding to `err`, then
- ▷ the formatted message, if non empty (same usage as printf), then
- ▷ a newline.

Usage:

```
err = myfunc();
if (err != NPIC_SUCCESS) return NpicError ("myfunc", err, "");

err = loadfile (filename);
if (err != NPIC_SUCCESS) return NpicError ("loadfile", err, " in \"%s\"", filename);
```

1.7.2 Computation time

Sources: misc.h , misc.c .

• **NpicGetTime**

```
double NpicGetTime ();
```

Return time since Epoch (1970, january the 1st at 00:00) in *seconds.microseconds* .

Usage:

```
double time1, time2;

time1 = NpicGetTime ();
...
time2 = NpicGetTime ();
printf ("Computation time : %.6lf s\n", time2-time1);
```

1.7.3 Text properties

Sources: props.h , props.c , image_prop.h , image_prop.c , mask_prop.h , mask_prop.c .

A *property* is a couple (`key`, `val`) where `key` and `val` are strings. A *property list* is a list of properties where each `key` is unique.

The `key` and `val` are not limited in size neither in number; but they must be 7-bit ANSI strings, to avoid internationalization problems. More precisely :

- ▷ A `key` is valid if it is a non-empty string only composed of chars in `[33,126]` and different from `':'`. In particular, chars `'\t'`, `' '` and `'\n'` are forbidden. This is checked with `NpicPropKeyIsValid`.
- ▷ A `val` is valid if it is a possibly empty string only composed of chars in `[32,126]` or a `'\t'`. In particular, char `'\n'` is forbidden, `' '` and `':'` are allowed. This is checked with `NpicPropValIsValid`.

To manage a list of properties, declare `Npic_props props`, initialize with `NpicInitProps (&props)`, then use one of the functions `NpicAddProp`, `NpicSupprProp`, `NpicGetProp`, `NpicHasProp`, `NpicPrintProps`, and finally, free with `NpicFreeProps (&props)`.

Each image and each mask has its own property list, which is automatically initialized at creation and freed at destruction. For images use the fonctions `NpicImage...` ; for masks use the fonctions `NpicMask...` These functions do nothing if the object is not ok.

• **NpicInitProps**

```
void NpicInitProps (Npic_props *props);
```

Initialize `Npic_props` structure. Silent.

• **Npic*FreeProps**

```
void NpicFreeProps (Npic_props *props);
void NpicImageFreeProps (Npic_image *m);
void NpicMaskFreeProps (Npic_mask *m);
```

Free `Npic_props` structure. Silent.

• **Npic*CopyProps**

```
int NpicCopyProps (Npic_props *dest, Npic_props *src);
int NpicImageCopyProps (Npic_image *dest, Npic_image *src);
int NpicMaskCopyProps (Npic_mask *dest, Npic_mask *src);
```

Copy properties from `src` to `dest` ; free `dest` before if non empty. On success return `NPIC_SUCCESS`, else error code. Silent.

• **NpicProp*IsValid**

```
int NpicPropKeyIsValid (const char *key);
int NpicPropValIsValid (const char *val);
```

Test if a `key` or a `val` is valid. Return 1 if valid, else 0. Silent.

• **Npic*AddProp**

```
int NpicAddProp (Npic_props *props, const char *key, const char *val);
int NpicImageAddProp (Npic_image *np, const char *key, const char *val);
int NpicMaskAddProp (Npic_mask *m, const char *key, const char *val);
```

Insert a property (`key`, `val`) in properties list (only if `key` and `val` are valid, else they are rejected). If `key` is already in list, the `val` is overwritten. The list is sorted by `key`, to allow fast dichotomic search. The list is enlarged if necessary. On success return `NPIC_SUCCESS`, else error code. Verbose.

• **Npic*SupprProp**

```
int NpicSupprProp (Npic_props *props, const char *key);
int NpicImageSupprProp (Npic_image *np, const char *key);
int NpicMaskSupprProp (Npic_mask *m, const char *key);
```

Remove a property (`key`, `val`) from properties list. The list is kept sorted, without hole. The list is shortened if necessary. On success return `NPIC_SUCCESS`, else error code. Verbose.

• **Npic*GetProp**

```
const char *NpicGetProp (Npic_props *props, const char *key);
const char *NpicImageGetProp (Npic_image *np, const char *key);
const char *NpicMaskGetProp (Npic_mask *m, const char *key);
```

Search a property from its `key`. Return property value if found, else `""`. Silent. To know if a property is set, use `NpicHasProp`.

• **Npic*GetPropD**

```
const char *NpicGetPropD (Npic_props *props, const char *key, const char *default_val);
const char *NpicImageGetPropD (Npic_image *np, const char *key, const char *default_val
const char *NpicMaskGetPropD (Npic_mask *m, const char *key, const char *default_val);
```

Same as `NpicGetProp`, except that it returns `default_val` if property is not found.

• **Npic*HasProp**

```
int NpicHasProp (Npic_props *props, const char *key);
int NpicImageHasProp (Npic_image *np, const char *key);
int NpicMaskHasProp (Npic_mask *m, const char *key);
```

Search a property from its `key`. Return 1 if found, else 0. Silent.

• **Npic*PrintProps**

```
void NpicPrintProps (Npic_props *props);
void NpicImagePrintProps (Npic_image *np);
void NpicMaskPrintProps (Npic_mask *m);
```

Print all properties.

As an example, see test-props.c

Chapter 2

Npic Types

2.1 Pixels

Sources: pixels.h

• Basic types

`Npic_sint8` and `Npic_uint8` are signed and unsigned 8 bits integers (generally defined as `char`).

`Npic_sint16` and `Npic_uint16` are signed and unsigned 16 bits integers (generally defined as `short`).

`Npic_sint32` and `Npic_uint32` are signed and unsigned 32 bits integers. On 32 bits systems, they are defined as `long`; on 64 bits systems, they are defined as `int`. To printf such an integer in a portable way, use `%"NPIC_PL"` instead of `%d` (the macro `NPIC_PL` is defined as `ld` or `d`).

• Byte swap

The macros `NPIC_BSWAP16(x)`, `NPIC_BSWAP32(x)`, `NPIC_BSWAP64(x)` swap the bytes on integers or floating numbers.

• Pixels definition

The pixel types are defined as follows:

```
typedef Npic_uint8   Npic_c;
typedef Npic_sint16  Npic_s;
typedef Npic_sint32  Npic_l;
typedef double       Npic_d;
```

• Quadri-color pixels

The type `Npic_q` is defined as follows, with alternatives:

```
typedef struct { Npic_uint16 a, b, c, d ; } Npic_q;
typedef struct { Npic_uint16 r, g, b, a ; } Npic_rgba;
```

```
typedef struct { Npic_uint16 c, m, y, k ; } Npic_cmyk;
typedef struct { Npic_uint16 h, s, v, a ; } Npic_hsva;
```

2.2 Images

Sources: images.h

• Npic_image

The `Npic_image` type is a union of structs :

```
typedef union {
    int              type;
    Npic_image_gen   gen;
    Npic_image_2c    im_2c;
    Npic_image_2l    im_2l;
    Npic_image_2d    im_2d;
    Npic_image_2q    im_2q;
    Npic_image_3c    im_3c;
    Npic_image_3l    im_3l;
    Npic_image_3d    im_3d;
    Npic_image_3q    im_3q;
    Npic_image_4c    im_4c;
    Npic_image_4l    im_4l;
    Npic_image_4d    im_4d;
    Npic_image_4q    im_4q;
    Npic_image_5c    im_5c;
    Npic_image_5l    im_5l;
    Npic_image_5d    im_5d;
    Npic_image_5q    im_5q;
    Npic_image_6c    im_6c;
    Npic_image_6l    im_6l;
    Npic_image_6d    im_6d;
    Npic_image_6q    im_6q;
} Npic_image;
```

• Nature

The field `type` is one of: `NPIC_IMAGE_2C`, `NPIC_IMAGE_2L`, `NPIC_IMAGE_2D`, `NPIC_IMAGE_2Q`, `NPIC_IMAGE_3C`, `NPIC_IMAGE_3L`, `NPIC_IMAGE_3D`, `NPIC_IMAGE_3Q`, `NPIC_IMAGE_4C`, `NPIC_IMAGE_4L`, `NPIC_IMAGE_4D`, `NPIC_IMAGE_4Q`, `NPIC_IMAGE_5C`, `NPIC_IMAGE_5L`, `NPIC_IMAGE_5D`, `NPIC_IMAGE_5Q`, `NPIC_IMAGE_6C`, `NPIC_IMAGE_6L`, `NPIC_IMAGE_6D`, `NPIC_IMAGE_6Q`,
where 2,3,4,5,6 stands for the dimension, and the letter stands for the pixel type: C for `Npic_c`, L for `Npic_l`, D for `Npic_d`, Q for `Npic_q`.

• Generic access

The field `gen` is a generic field for accessing to the struct fields, which is possible because

all the image structs are strictly identical in memory. Some fields are protected with a '_', they should *not* be used.

```
typedef struct {
    int  type,                                  /* Nature : NPIC_IMAGE_2C, ..   */
         ok,                                    /* status of image              */
         pixsize,                               /* Memory size of one pixel     */
         dim,                                   /* Dimension : 2 .. 6           */
         xbor, ybor, zbor, tbor, sbor, rbor,    /* Width of the borders  : >= 0 */
         xmax, ymax, zmax, tmax, smax, rmax,    /* Width, height, ..     : >  0 */
         xtot, ytot, ztot, ttot, stot, rtot;    /* Total width, ..       : >  0 */
    size_t pixtot;                              /* Total number of pixels: > 0  */
    void **vy_, **vz_, **vt_, **vs_, **vr_;     /* Base addr of lines, plans .. */
    void *vec_,                                 /* Linear array of pixels       */
         *pix_;         /* Access [y][x] or [z][y][x] or .. [r][s][t][z][y][x]  */
    Npic_props props;   /* To store a list of text properties, see props.h      */
} Npic_image_gen;
```

2.3 Masks

Sources: masks.h

• **Npic_mask**

The `Npic_mask` type is a union of structs :

```
typedef union {
    int            type;
    Npic_mask_gen  gen;
    Npic_mask_2l   dm_2l;
    Npic_mask_2d   dm_2d;
    Npic_mask_3l   dm_3l;
    Npic_mask_3d   dm_3d;
    Npic_mask_4l   dm_4l;
    Npic_mask_4d   dm_4d;
    Npic_mask_5l   dm_5l;
    Npic_mask_5d   dm_5d;
    Npic_mask_6l   dm_6l;
    Npic_mask_6d   dm_6d;
} Npic_mask;
```

• **Nature**

The field `type` is one of: `NPIC_MASK_2L`, `NPIC_MASK_2D`, `NPIC_MASK_3L`, `NPIC_MASK_3D`, `NPIC_MASK_4L`, `NPIC_MASK_4D`, `NPIC_MASK_5L`, `NPIC_MASK_5D`, `NPIC_MASK_6L`, `NPIC_MASK_6D`, where 2,3,4,5,6 stands for dimension, L stands for `Npic_l`, D for `Npic_d`.

• **Generic access**

The field `gen` is a generic field for accessing to the struct fields, which is possible because all the mask structs are strictly identical in memory.

```
typedef struct {
    int   type,          /* Nature : NPIC_MASK_2L, ..                  */
          ok,            /* Status of mask                             */
          size,          /* Memory size of one vector                  */
          nb,            /* Current number of vectors                  */
          tot,           /* Total number of allocated vectors          */
          step;          /* Increment for realloc                      */
    void  *v;            /* Allocated list of weightings               */
    Npic_props props;    /* To store a list of properties, see props.h */
} Npic_mask_gen;
```

• **Weightings**

A weighting is a vector `(..,y,x)` and a value `h` (a weight, a radius, etc).

```
typedef struct { int y, x;             Npic_l  h; } Npic_weighting_2l;
typedef struct { int y, x;             Npic_d  h; } Npic_weighting_2d;
typedef struct { int z, y, x;          Npic_l  h; } Npic_weighting_3l;
typedef struct { int z, y, x;          Npic_d  h; } Npic_weighting_3d;
typedef struct { int t, z, y, x;       Npic_l  h; } Npic_weighting_4l;
typedef struct { int t, z, y, x;       Npic_d  h; } Npic_weighting_4d;
typedef struct { int s, t, z, y, x;    Npic_l  h; } Npic_weighting_5l;
typedef struct { int s, t, z, y, x;    Npic_d  h; } Npic_weighting_5d;
typedef struct { int r, s, t, z, y, x; Npic_l  h; } Npic_weighting_6l;
typedef struct { int r, s, t, z, y, x; Npic_d  h; } Npic_weighting_6d;
```

2.4 Miscellaneous

2.4.1 G-symmetries

Sources: mask_gsym.h

The struct `Npic_gsym` is used for computing the G-symmetries.

```
#define NPIC_GSYM_MAX 20
typedef struct {
   int dim,                /* Dimension and coordinates */
       x[NPIC_GSYM_MAX],   /* (x[1], x[2], ..., x[dim]) */
       half,               /* Compute 1: half, 0: full mask */
       dir[NPIC_GSYM_MAX], /* Direction and position for */
       pos[NPIC_GSYM_MAX], /* Johnson's permutation algorithm */
       k, kmax,            /* Number of sign combination */
       bin_mask;           /* Binary mask to forbid combinations */
} Npic_gsym;
```

2.4.2 Vector

Sources: misc.h

The struct `Npic_vec` stores the coordinates of a vector. The use of fields `z, t, s, r` is dimension dependent.

```
typedef struct { int x, y, z, t, s, r; } Npic_vec;
```

2.4.3 Lut for medial axis

Sources: mlut_remy.h

The struct `Npic_Lut` is used for storing the look-up table in medial axis extraction.

```
typedef struct {
    int col_nb,     /* Number of allocated columns */
        col_tot,    /* size of vector of column addresses */
        col_len;    /* size of one column */
    Npic_l **v;
} Npic_Lut;
```

2.4.4 Text Properties

Sources: props.h

The followings structs are used for storing text properties.

```
typedef struct { char *key, *val; } Npic_atom;
typedef struct { int nb, tot; Npic_atom *list; } Npic_props;
```

Chapter 3

Npic File Formats

3.1 Image file format NPZ

The NPZ file format (with `.npz` extension) is the proposed format to read and save Npic images, whatever the dimension, the pixel type and the endianness. The beginning of the file is a text header describing the content (use Unix command `head` to watch it) and the text properties, the next is the bitmap compressed with gzip. The library file functions are described in section 1.3.2 .

The NPZ file format can store text properties. A text property is a couple of strings `"keyword: value"` on a single line. Properties are not limited in number neither in size, see section 1.7.3 .

Here is an exemple with the header of a 3-d image having one text-property:

```
NPZ-11
# Image file generated by libNpic - Mon Dec  3 16:24:17 2007
TYPE NPIC_IMAGE_3C
COMP GZIP
XMAX 50
YMAX 50
ZMAX 50
PROP Alpha-Color: 0
DATA
```

The command-line tool npic-new allows to create an image; npic-draw draws pixels in it.

Format specification

The format file begins with a text header and ends with the binary compressed datas. The header mandatory starts by `NPZ-11`. Then, in any order, you can find :

- ▷ Comments, starting by `#` at any position in the line, ending with the line.
- ▷ The word `TYPE` then a *blank* (a sequence of spaces, tabs and carriage returns), then one of the words : `NPIC_IMAGE_2C`, `NPIC_IMAGE_2L`, `NPIC_IMAGE_2D`, `NPIC_IMAGE_2Q`, `NPIC_IMAGE_3C`, `NPIC_IMAGE_3L`, `NPIC_IMAGE_3D`, `NPIC_IMAGE_3Q`, `NPIC_IMAGE_4C`, `NPIC_IMAGE_4L`, `NPIC_IMAGE_4D`, `NPIC_IMAGE_4Q`, `NPIC_IMAGE_5C`, `NPIC_IMAGE_5L`, `NPIC_IMAGE_5D`, `NPIC_IMAGE_5Q`, `NPIC_IMAGE_6C`, `NPIC_IMAGE_6L`, `NPIC_IMAGE_6D`, `NPIC_IMAGE_6Q`, (`2,3,4,5,6` stands

for the dimension, `C` for unsigned int8, `L` for signed int32, `D` for double, `Q` (quadricolor) for 4 unsigned int16); finally a blank.

- ▷ The word `COMP` followed by the compression mode and a blank; currently, only `GZIP` is accepted.
- ▷ The word `XMAX` followed by an integer (the image side length in x) and a blank. The same with `YMAX`, `ZMAX`, `TMAX`, `SMAX` and `RMAX`, depending on dimension.
- ▷ The word `PROP`, followed by a blank and a text property, which is a keyword, a ":", a space, then the value which is the rest of the line. For more details, see section 1.7.3 . Note that no property is currently reserved.

Giving `TYPE`, `COMP` and image side lengths is mandatory. The text header ends with `DATA`, followed by exactly one char.

The second part is the binary datas compressed through a pipe to gzip (or gunzip when reading). If you suppress the text header, you can then uncompress the resulting file with gzip.

The binary datas (before compression) starts with a magic value depending on pixel type (`C` : no magic value ; `Q` : 0x1234 ; `L` : 0x12345678 ; `D` : 1.2345678987654321e-279). The intent is to test if the endianness of the source program and the destination program is the same. If not, every int16 or int32 or double will be byte-swapped on reading. This way, no care about endianness is taken on writing, only on reading if necessary.

Following the magic value, come the raw datas, which are binary pixels values in scan line order (`for r, for s, for t, for z, for y, for x`).

3.2 Mask file format NMASK

The Npic file format NMASK allows to store a mask in a file (with extension `.nmask` or `.nmask.gz`). The mask functions are described in section 1.4 , the file functions in 1.4.8 , and the mask type in 2.3 .

A mask is a list of vectors, each vector being associated to a weight (a local distance, an appearance radius, etc). The vectors must have integral coordinates, in dimension 2 to 6. The weights can be signed int32 or double.
Text properties can also be stored in the file. A text property is a couple of strings `"keyword: value"` on a single line. Properties are not limited in number neither in size, see section 1.7.3 . Some properties are already reserved, see below, but any other properties can be added as you need.

Here is an exemple with the 2D mask of the distance chamfer 5-7-11 :

```
NMASK-11
# Mask file generated by libNpic - Thu Jan  6 17:06:19 2011
TYPE NPIC_MASK_2L
PROP AddSym: GSymToHalfMask
PROP DistanceType: Weighted
PROP Nature: Distance
#  y    x      h
DATA
```

```
0    1      5
1    1      7
1    2     11
```

The command-line tool npic-mask allows to generate or manipulate masks; npic-wdt computes weigthed distance transforms.

Format specification

The format is a full text file beginning with a header and ending with the datas. The header mandatory starts by `NMASK-11`. Then, in any order, you can find :

- ▷ Comments, starting by `#` at any position in the line, ending with the line.
- ▷ The word `TYPE` then a *blank* (a sequence of spaces, tabs and carriage returns), then one of the words : `NPIC_MASK_2L`, `NPIC_MASK_2D`, `NPIC_MASK_3L`, `NPIC_MASK_3D`, `NPIC_MASK_4L`, `NPIC_MASK_4D`, `NPIC_MASK_5L`, `NPIC_MASK_5D`, `NPIC_MASK_6L`, `NPIC_MASK_6D`, (`2,3,4,5,6` stands for the dimension, `L` for signed int32, `D` for double); finally a blank.
- ▷ The word `PROP`, followed by a blank and a text property, which is a keyword, a ":", a space, then the value which is the rest of the line. For more details, see section 1.7.3 .

Giving the `TYPE` of the mask is mandatory. The header ends with `DATA\n`, followed by the list of vectors coordinates and weights. Depending on the dimension, each line stores "y x h" or "z y x h" or "t z y x h" or "s t z y x h" or "r s t z y x h" (h is the weight). The order of vectors is kept in the file.

Reserved properties

Some text properties are specified :

- ▷ The property `Nature` describes the nature of the masks: `Distance` for a half distance mask, `MedialAxis` for a medial axis test neighbourhood (sometimes denoted MLut in the literature), `Convolution` for a convolution mask; `Unknown` else.
- ▷ The property `AddSym` tells if vectors must be added after reading by making symmetries. Current supported value is `GSymToHalfMask` to make all grid symmetries from Generator to the half mask, or `None`.
- ▷ The property `DistanceType` is used to define the distance: `Weighted` (default), `SquaredEuclidean`, `CeilEuclidean`, `FloorEuclidean`. Note that for the Euclidean distances, the `DATA` part of the file is empty.
- ▷ The property `DistanceMask` is used in medial axis masks, to store the name of the distance mask.
- ▷ The property `RVerified` is used in medial axis masks, to store the largest verified radius for that distance.