Image Toolbox Manual

Chapter 3

Library Function Descriptions

Chapter 3 - Library Function Descriptions

A. Introduction

This chapter contains the man3 pages for the function calls in the kimage_proc library that is contained in the Image toolbox. There are two categories of functions in the library, each presented in its own section. The first section contains the library calls that are associated with the kroutines in the Image toolbox, and the second section contains utility functions. The man3 pages in each section are alphabetically ordered by function name.

ALL programs that utilize these function calls MUST include the statement:

```
#include <image.h>
```

A.1. The kimage_proc Library

The kimage_proc library (libkip.a) contains image processing operators such as frequency filter generation, color compression, data generation, image filtering, postscript conversion, rotation, and geometric warping. All kimage_proc functions are written to operate on width-height planes of the polymorphic data model. If the depth, time, or elements dimensions of data object are greater than one, the operation is repeated for each width-height plane. The kimage_proc operators are implemented using the polymorphic data services, which is a part of the Khoros 2.0 kappserv library.

A.2. Side Effects

The function calls in kimage_proc should have no side effects on the source objects passed into them, and should only affect those segments and attributes of the destination object that the operation *needs* to modify. For example, if a destination object is passed in that already has value, mask, and geometry data, and a multiplication operation is performed on the source value data, the destination mask and geometry data will not be modified, and will remain intact.

Each function call frees all memory that has been allocated within the call, and closes all temporary objects that were opened within the call. The function call will *not* free memory or close objects that were passed into it.

A.3. DISCLAIMER — WARNING

The kimage_proc library is in its first prototype stage, therefore, the behavior and API of these functions may change in the future.

A.4. On-Line Access to man3 Pages

Any library function man3 page can be accessed by typing the following command.

```
% kman <function_name>
```

The keywords (-k) option in kman allows the user to provide a keyword string for searching. When invoked with the *-k* option, kman will perform the search and provide a list of the names and short descriptions of functions found by the search.

```
% kman -k "keyword string"
```

B. kimage_proc Library Functions

B.1. kroutine Function Calls in the kimage_proc Library

B.1.1. lifilt_design() — *generate 2-dimensional frequency filter*

Synopsis

```
int lifilt_design (
     int  type,
     int  brand,
     int  order,
     double   pixel_height,
     double   pixel_width,
     int  image_height,
     int  image_width,
     double   lower_cutoff,
     double   upper_cutoff,
     int  centered,
     kobject freq_image)
```

Input Arguments

`type`
: type of filter.

`brand`
: brand of filter.

`order`
: filter order.

`pixel_height`
: vertical size of a pixel

`pixel_width`
: horizontal size of a pixel

`image_height`
: height of the image.

`image_width`
: width of the image.

`lower_cutoff`
: lower cutoff for band pass or band stop, and cutoff for high pass filters.

`upper_cutoff`
: upper cutoff for band pass or band stop, and cutoff for low pass filters.

`centered`
: if TRUE, it puts low frequencies in the center of the image. Otherwise it puts them in the corners.

Output Arguments

`freq_image`
: An opened kobject to store the frequency image in

Returns

TRUE (1) on success, FALSE (0) on failure

Description

This routine generates a spatial frequency image of real values that represents the response of the filter type and brand that was specified. The types it supports are Low Pass, High Pass, Band Pass, and Band Stop filters. The generator function brands are Ideal, Butterworth, Exponential, Chebyshev I, and Chebyshev II. All output is radially semmetric about the origin of the image, and the origin can be either the center of the image or the top left corner of the image. Cutoff frequencies are normalized between 0 and 0.5 where 0 is DC, and 0.5 is the maximum circle/elipse that will cover the image. Image height and width do not have to be the same, but pixel distance to the origin will be normalized by image height and width dimensions. For example, if you have an image of size 256x128 with the origin at the center of the image, then the pixel at 0,128 and 64,0 will both be normalized to 0.5. This will produce filters that look like an elipse with the major axis in the direction of the largest image dimension, and the minor axis in the smallest dimension.

Restrictions

The parameters pixel_height and pixel_width are not yet supported by lifilt_design. Currently, only a value of 1 is accepted for these parameters.

B.1.2. ligauss_func() — *generate 2D gaussian function data*

Synopsis

```
int ligauss_func(kobject param_obj,
                 int rows, int cols,
                 int xpeak, int ypeak,
                 double xvar, double yvar,
                 double coeff, double val,
                 int norm, int type,
                 kobject dst_obj)
```

Input Arguments

`param_obj`
: object containing multiple gaussina parameters

`rows`
: number of rows in output object

`cols`
number of columns in output object
`xpeak`
x coordinate of function peak value
`ypeak`
y coordinate of function peak value
`xvar`
variance along the x direction
`yvar`
variance along the y direction
`coeff`
correlation coefficient between the 2 dimensions
`val`
peak value of the gaussian function
`norm`
flag indicating normalisation, if TRUE multiple gaussian amplitude is normalised
`type`
datatype of the output object

Output Arguments

`dst_obj`
destination object contain gaussian function

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

library routine for igauss_func. ligauss_func creates a 2D gaussian function value data set containing one or many gaussian functions.For each gaussian function generated, 6 parameters are required: (x,y) coordinate of the peak, variance along x&y direction, correlation coefficient between the two variables of the 2D gaussian function and the peak value of the gaussian function. For generating more than one gaussian function this information must be stored in param_obj along its bands, organised in the order described above. If param_obj is NULL it is assumed that only one gaussian is to be generated. The parameters of that gaussian are passed in directly. For the multiple gaussian case the amplitude values are ignored if norm is TRUE. The output object, dst_obj has size rows*cols*1 and datatype given by type. Complex data types are not supported for ligauss_func.

B.1.3. ligeowarp() — *perform direct bilinear warping in the WxH plane*

Synopsis

```
int ligeowarp(
kobject in_obj,
double a00,
double a01,
```

```
double a10,
double a11,
double b00,
double b01,
double b10,
double b11,
int planes,
kobject out_obj)
```

Input Arguments

`in_obj`
: input object to be processed

`a00`
: constant term coeff in W mapping function

`a01`
: w constant term coeff in W mapping function

`a10`
: h constant term coeff in W mapping function

`a11`
: wh constant term coeff in W mapping function

`b00`
: constant term coeff in H mapping function

`b01`
: w constant term coeff in H mapping function

`b10`
: h constant term coeff in H mapping function

`b11`
: wh constant term coeff in H mapping function

`planes`
: if non-zero, process date by full planes instead of prisms. Can be MUCH faster by planes than prisms, but is less well adapted for large data sets.

Output Arguments

`out_obj`
: output object

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

perform direct bilinear warping in the WxH plane. This is a reasonable way to register one image to another provided that the distortion is not too severe. Output pixel values are calculated using bilinear interpolation.

B.1.4. ligradient() — *perform image differentiation using gradient operator*

Synopsis

```
int ligradient (
          kobject src,
                 int gradient_type,
                 int operation_type,
                 kobject dest )
```

Input Arguments

`src`
the source data object to be processed

`gradient_type`
type of gradient calculation

`operation_type`
absolute values (default) or squares and square root operations

Output Arguments

`dest`
the destination data object

Returns

TRUE (1) on success, FALSE (0) on failure

Description

takes an image and performs the differentiation (high pass spatial filtering or edge enhancement) operation by using standard gradient operators. The standard gradient operators supported by ligradient are: Robert's, Sobel, Prewitt and Isotropic). The Robert's operators use a 2x2 kernel, whereas the other three use a 3x3 kernel. Each operator uses two templates to enhance edges in the horizontal and vertical directions only. For edge enhancements in other directions other operators should be used.

There are two approximations to the true gradient. The user can select between calculating the magnitude of the gradient by using squares and square roots (Euclidean distance evaluation), or by using absolute differences. The latter is computational faster.

Mask data is ignored.

B.1.5. limedian() — *perform median filtering*

Synopsis

```
int limedian(kobject in_obj, int wsize, int hsize, int reps, kobject out_obj)
```

Input Arguments

`in_obj`
: input object to be processed

`wsize`
: width of window in pixels

`hsize`
: height of window in pixels

`reps`
: number of repetitions of filter to perform

Output Arguments

`out_obj`
: output object

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

for each WxH window in the image, replace the pixel at (W/2)x(H/2) (integer arithmetic) with the median of the values in the window. The median is computed by sorting the values in the window with quicksort and extracting the middle value as the median. This is not the fastest approach for BYTE data where a histogram updating method could be used. It is, however, extensible to higher data types including floating point data.

If a map is present, the data is mapped prior to operation. Mask data is ignored.

This implementation processes the image data by full planes.

Boundaries are handled by zero padding.

Restrictions

Complex data types are not handled at all.

B.1.6. lipostscr() — *convert input image object to postscript*

Synopsis

```
int lipostscr(
    kfile *file,
    kobject obj,
    double width,
    double height,
    double xoffset,
    double yoffset,
```

```
int wflg,
int hflg,
int xflg,
int yflg,
int landscape,
int eps,
int showpage)
```

Input Arguments

`obj`
    Object to be output in postscript form
`width`
    Desired width in inches of the printed image
`height`
    Desired height of printed image in inches
`xoffset`
    Width offset in inches from bottom left corner of printed page
`yoffset`
    Height offset in inches from bottom left corner of printed page
`wflg`
    if TRUE, use width input parameter
`hflg`
    if TRUE, use height input parameter
`xflg`
    if TRUE, use xoffset input parameter
`yflg`
    if TRUE, use yoffset input parameter
`landscape`
    if TRUE, output as landscape, otherwise portrait
`eps`
    if TRUE, output as encapsulated postscrip, otherwise output as
`showpage`
    if TRUE, issue showpage at end of output

Output Arguments

`file`
    output postscript file

Returns

TRUE (1) on success, FALSE (0) on failure

Description

lipostscr formats an image for output on a Postscript printer. The width and height are specified in inches. The X and Y offset specifies the location of the lower left corner of the image from the lower left corner of the page.

lipostscr will generate either a binary, grayscale, truecolor image from the input object. If the data set contains a map, then the first three columns (assuming that there are at least three) will be interpreted a the red, green, and blue components and will be used to construct a truecolor postscript image by mapping the value data through the map. If a map exists, but it does not have three or more columns, then the existing columns will be padded out to three columns with 0's.

If the data set does not contain a map, then if its data type is KBIT, it will generate a bitmapped black and white postscript. Otherwise, the data set will be cast to unsigned byte and a grayscale image will be generated.

This program can be used to generate either Postscript or Encapsulated Postscript images in either Landscape or Portrait mode. The -eps flag is used to specify that encapsulated postscript should be output.

If you only specify the width, or the height, the other dimension will be calculated, so that you will get a postscript image which reflects the true dimensions of the image.

B.1.7. lirotate() — *rotate image plane by arbitrary angle about an arbitrary point*

Synopsis

```
int lirotate(
    kobject in_obj,
    double wctr,
    double hctr,
    int resize,
    int planes,
    double angle,
    kobject out_obj)
```

Input Arguments

`in_obj`
: input object to be processed

`wctr`
: W coordintae of center of rotation

`hctr`
: H coordintae of center of rotation

`resize`
: if non-zero, resize output to fit data

`planes`
: if non-zero, process date by full planes instead of prisms. Can be MUCH faster by planes than prisms, but is less well adapted for large data sets.

`angle`
: rotation angle in degrees. Positive is CW.

Output Arguments

`out_obj`
: output object

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

rotate image plane about an arbitrary point and an aribitrary angle. Pixel values are obtained by bilinear interpolation.

B.1.8. litexture() — *texture feature extraction using LAW metrics*

Synopsis

```
int litexture(
        kobject in_obj,
        int L5L5,
        int L5E5,
        int L5S5,
        int L5W5,
        int L5R5,
        int E5L5,
        int E5E5,
        int E5S5,
        int E5W5,
        int E5R5,
        int S5L5,
        int S5E5,
        int S5S5,
        int S5W5,
        int S5R5,
        int W5L5,
        int W5E5,
        int W5S5,
        int W5W5,
        int W5R5,
        int R5L5,
        int R5E5,
        int R5S5,
        int R5W5,
        int R5R5,
        kobject out_obj)
```

Returns

TRUE (1) on success, FALSE (0) on failure

Description

lktexture will extract texture features from an image by convolving the original image with a spatial filter. The spatial filters are comprised of 5 x 5 kernals derived from center-weighted vectors defined by the LAWS texture measure.

The five center-weighted vectors are:

L5 = [1 4 6 4 1]
E5 = [-1 -2 0 2 1]
S5 = [-1 0 2 0 -1]
W5 = [-1 2 0 -2 1]
R5 = [1 -4 6 -4 1]

Each 5 x 5 kernal is derived from multiplying a particular combination of two of the above vectors. This results in 25 possible 5 x 5 kernals.

If the input object elements dimension is greater than 1 then the results obtained by applying all texture kernels to the first band are stored by planes in the output object, followed by the results obtained by applying all texture kernels to the second band, etc.

lklinearop() is called to convolve the input image with the selected kernels. The result are stored as a WxH planes along the E direction in the output object.

Restrictions

lktexture will operate on input objects of any data storage type.

The output object will be of type KLONG, KDOUBLE, or KDCOMPLEX, determined by the data type of the input object. If the input object is of data type KBYTE, KUBYTE, KSHORT, or KUSHORT, then it is converted up to an KLONG image. If the input is of type KFLOAT or KDOUBLE, then the result will be KDOUBLE. Any complex input type is converted to type KDCOMPLEX.

B.2. Utility Functions in the kimage_proc Library

B.2.1. gauss_gen() — *gaussian generator for a line of data*

Synopsis

```
int gauss_gen(
    int xpeak,
    int ypeak,
    double xvar,
    double yvar,
    double corr_coeff,
    double ampl,
```

```
        int norm,
        int size,
        int start_xpos,
        int ypos,
        int flag,
        double **data)
```

Input Arguments

`xpeak`
: x location of peak of gaussian function

`ypeak`
: y location of peak of gaussian function

`xvar`
: variance along x dimension

`yvar`
: variance along y dimension

`corr_coeff`
: correlation coefficient between 2 dimensions

`ampl`
: amplitude of the gaussian function

`norm`
: normalization flag, if set normalize

`size`
: size of the output data

`start_xpos`
: starting position of x coordinate

`ypos`
: y position of gaussian function

`flag`
: add flag, if set values are added to existing values

Output Arguments

`data`
: array containing gaussian data

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

Generates a 2D gaussian function for a line in the image.

Restrictions

Restrictions on data or input as applicable

Table of Contents

This page left intentionally blank