**VisionX V4 Simple Interface Tutorial**

In VisionX programs, files are read into image structures to facilitate easy programming. A file structure is declared by a vfstruct statement, for example:

Vfstruct (im);
Vfstruct (om);

Note, only one structure may be declared in a statement.

Consider we wish to read a byte image and threshold it a grey level of 50. That is, the output image of the program will contain 0 where the input image is less than 50 and will contain 255 where the input image greater than 50. Further, let the input image file be called infile.vx and the output image be called outfile.vx.

The following program will accomplish this:

Example 1. A minimal program

```
#include "VisXV4.h"    /* VisionX structure include file       */
#include "Vutil.h"     /* VisionX utility header files         */

int main(int argc, char** argv)
{
Vfstruct (im);
Vfstruct (om);
int x, y;
    Vfread( &im, "infile.vx");
    Vfnewim (&om, im.type, im.bbx, im.chan);
    for ( y = im.ylo; y <= im.yhi; y++) {
       for ( x = im.xlo; x <= im.xhi; x++) {
          if ( im.u[y][x] >= 50 ) {
              om.u[y][x] = 255;
          } else {
              om.u[y][x] = 0;
          }
       }
    }
    Vfwrite( &om, "outfile.vx");
exit(0);
}
```

Notes:

1. All programs in VisionX should start with the two include statements “VisXV4.h” and “VXutil.h” as shown above. Note, you do NOT need to include that standard C programming language files <stdio.h> and <math.h> these have already been included.
2. Vfread reads the file inimage.vx and builds the internal image data array in the Vfstructure im.
3. Vmakeimage creates a image (set to zero) with the same parameters as the image im. Parameters specify the image type (byte in this case), bounding box and number of channels.
4. A double loop is used to compute the function on the image im and store the result in the image om. Some features of this loop are a follows:
   a. Not the outer loop is associated with the y (vertical) direction and the inner loop is associated with the x (horizontal direction). This may appear to be counter intuitive but provides for the most efficient (fastest) implementation.
   b. The image indicies do not necessarily start at 0 ro 1. The lowest image location is (im.xlo, im.ylo)
   c. Since the highest legal index values are im.xhi and im.yhi the condition in the loop expression is “>=” rather than just “>” which is the more typical value used in C program loops.
   d. For the image elements you must supply the image type explicitly since c programming is not polymorphic. That is, for the expression im.u[i][j] the .u is required to specify that we are refereeing to an image of type unsigned byte.
5. The Vwrite statement writes the image in om to the file “outfile.vx

Essential members of the Vfstruct that you must know:

| | |
|---|---|
| .bbx | The bounding box for the image (float[6]) |
| .type | The base type of the image pixels; see next example for a list |
| .chan | The number of channels in the image (e.g., color has 3) |
| .xlo | The lowest defined x index |
| .xhi | The highest defined x index |
| .ylo | The lowest defined y index |
| .yhi | The highest defined y index |

Next we will consider a version of this program that uses some more advanced VisionX programming features.

Example 2 is a program allows the image file names to be specified on the command line, allows the threshold to be specified on the command line, and also works on image files that contain more than one image.

Example 2 A program that process multiple images in a file

```
#include "VisXV4.h"    /* VisionX structure include file       */
#include "Vutil.h"     /* VisionX utility header files         */

VXparam_t par[] =      /* command line structure               */
{
{    "if=",    0, " input file  vthresh: threshold images"},
{    "of=",    0, " output file "},
{    "th=",    0, " Threshold value (default 50)"},
{     0,       0,    0}
};
#define  INIMAGE   par[0].val
#define  OUTIMAGE  par[1].val
#define  THRESHOLD par[2].val

int main(int argc, char** argv)
{
Vfstruct (im);
int x, y;
int thresh;

VXparse(&argc, &argv, par);  /* parse the command line         */
 if (THRESHOLD ) {
    thresh = atoi (THRESHOLD);
 } else {
    thresh = 50;
 }

 while ( Vfread( &im, INIMAGE) ) {
    if ( im.type != VX_PBYTE ) {
           fprintf (stderr, "error: image not byte type\n");
           exit (1);
    }
    for ( y = im.ylo; y <= im.yhi; y++) {
       for ( x = im.xlo; x <= im.xhi; x++) {
          im.u[y][x] = im.u[y][x] >= thresh ? 255 : 0;
       }
    }
    Vfwrite( &im, OUTIMAGE);
  }
exit(0);
}
```

Notes:

1. Command line parameters have now been added to this program.
2. To perform exactly the same function as the previous program one would need to give the command:

   vthresh if=infile.vx of=outfile.vx

   The “par” structure defines a total of three parameters if= for the input file name and of= for the output file name; the if= and of= prefixes are used by convention throughout the VisionX system. A third optional parameter th= is also specified which allows the image threshold to be set to any value (not just 50).
3. The VXparse function matches command line parameters where possible to the .val elements. Therefore, after parsing the command line specified above, the par .val elements would have the following contents:
   par[0].val “inimage.vx”
   par[1].val “outimage.vx
   par[2].val “”
   The #define statements in the program provide a convenient way to name the different par .val elements.
4. If the th= option is specified on the command line then the string given as the threshold will be converted to an integer value by the atoi function and assigned to thresh.
5. The program as written will correctly operate on images with any dimensions and any number of channels but will only operate on images of type unsigned byte. Therefore a check has been added to ensure that the input image file is of the correct type.
6. Since there are no dependencies in this simple program there is no real need to create an image structure for the result; it may be written back into the original image. Also a more succinct c language version or the if expression in the first program is used in the second program which performs the same function.

Table 1 base pixel types supported by VisionX and their Vfstruct names

| Vfstruct type Name | C-Type | Vfstruct Member | Description |
|---|---|---|---|
| VX_PBYTE | unsigned char | .u | most usual byte format |
| VX_PFLOAT | float char | .f | 32-bit floating point |
| VX_PDOUBLE | double | .d | 64-bit floating point |
| VX_PBIT | unsigned char | .b | 1-bit (packed 8/byte) |
| VX_PCHAR | char | .c | 8-bit signed integer |
| VX_PSHORT | short | .s | 16-bit signed integer |
| VX_PINT | int | .i | 32-bit signed integer |
| VX_PIDX | unsigned char | .u | 8-bit color index |

Example 3 A 3D image processing program example

```
/*************************************************************************/
/* v3mean   Compute 3x3x3 3D mean filter on byte images                  */
/*************************************************************************/
#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */

VXparam_t par[] =            /* command line structure               */
{
{    "if=",    0,   " input file  v3mean: compute local mean"},
{    "of=",    0,   " output file "},
{     0,       0,    0}
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val

int
main(argc, argv)
int argc;
char *argv[];
{
V3fstruct (im);
V3fstruct (tm);
int        x,y,z;              /* index counters                */
int        xx,yy,zz;           /* window index counters         */
int        sum;
    VXparse(&argc, &argv, par); /* parse the command line        */

    V3fread( &im, IVAL);       /* read 3D image                 */
    if ( im.type != VX_PBYTE || im.chan != 1) { /* check  format  */
       fprintf (stderr, "image not byte type or single channel\n");
       exit (1);
    }

    V3fembed(&tm, &im, 1,1,1,1,1,1); /* temp image copy with border */

    for (z = im.zlo; z <= im.zhi; z++) {/* for all pixels */
      for (y = im.ylo; y <= im.yhi; y++) {
        for (x = im.xlo; x <= im.xhi; x++) {
             sum = 0;
             for (zz = -1; zz <= 1; zz++) {/* compute the function */
               for (yy = -1; yy <= 1; yy++) {
                 for (xx = -1; xx <= 1; xx++) {
                     sum = sum + tm.u[z + zz][y + yy][x + xx];
                 }
               }
             }
             im.u[z][y][x] = sum/9;
        }
      }
   }
   V3fwrite (&im, OVAL);
   exit(0);
}
```

Notes:

1. The program, in general, is similar to 2D except that there is an additional image index (z);
2. Use V3fread, V3fwrite, V3newim and V3embed instead of the 2D versions. Also use V3fstruct instead of Vfstruct.
3. The 3D program reads in a whole image file and treats it as a single 3D image. Therefore, there is no need to loop for multiple images as in the previous example. Files containing multiple 3D images are currently not directly supported

The 3D image structure has two additional elements for indexing in the z dimension as shown below. The base pixel types are the same as for 2D structures.

Critical members of the V3fstruct that you must know:

| | |
|---|---|
| .bbx | The bounding box for the image (float[6]) |
| .type | The base type of the image pixels; see next example for a list |
| .chan | The number of channels in the image (e.g., color has 3) |
| .xlo | The lowest defined x index |
| .xhi | The highest defined x index |
| .ylo | The lowest defined y index |
| .yhi | The highest defined y index |
| **.zlo** | **The lowest defined z index** |
| **.zhi** | **The Highest defined z index** |

Example 4. A time-buffer Image Processing example:

```
*********************************************************************//*
vbmean  Compute local 3x3x3 mean using buffer method              */
/*********************************************************************/

#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */

VXparam_t par[] =            /* command line structure               */
{
{    "if=",    0,   " input file vbmean: compute local mean"},
{    "of=",    0,   " output file "},
{     0,       0,    0}
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val

int
main(argc, argv)
int argc;
char *argv[];
{
V3fstruct (im);
V3fstruct (tm);
int        x,y,z;           /* index counters                */
int        xx,yy,zz;        /* window index counters         */
int        sum;

    VXparse(&argc, &argv, par); /* parse the command line    */

    while (Vbfread( &im, IVAL, 3)) {
        if ( im.type != VX_PBYTE || im.chan != 1) { /* check  format  */
            fprintf (stderr, "image not byte type\n");
            exit (1);
        }
        V3fembed(&tm, &im, 1,1,1,1,0,0); /* temp image with border */
          z=1;
          for (y = im.ylo; y <= im.yhi; y++) {
            for (x = im.xlo; x <= im.xhi; x++) {
             sum = 0;
             for (zz = -1; zz <= 1; zz++) {/* compute the function */
               for (yy = -1; yy <= 1; yy++) {
                 for (xx = -1; xx <= 1; xx++) {
                     sum = sum + tm.u[z + zz][y + yy][x + xx];
                 }
               }
             }
             im.u[0][y][x] = sum/9;
            }
          }
       V3fwrite (&im, OVAL);
     }
    exit(0);
}
```

The time-buffer programming model is for situations such as movies where you want to do some temporal processing; i. e., compute a function on several image frames. The 3D model could address this problem if the file was small enough to read into memory. However movies may be too large to fit into memory. Furthermore, in real time applications you would like to process the first frames of the movie while data acquisition for later frames is still in progress. The buffer processing mode provides support for this application. It provides a buffer of the <n> most recent image frames; then when a Vbfread operation is performed the oldest frame is discarded and a new frame is read in. That is, the Vrfbuf operation maintains a time ordered buffer of the <n> most recent image frames in a 3D image structure where z=0 is the z index value of the oldest frame and z=n-1 is the index of the most recent frame.

The example program performs a 3x3x3 mean filter operation similar to Example 3.

Notes:

1. Vbfread is used for reading data instead of V3fread. The first time Vbfread is called it reads n (=3) images; Subsequent calls read one new image and discard the oldest image.
2. With a buffer size of 3 the oldest frame has z index 0 and the most recent frame has z index 2.
3. The oldest image (z=0) may be used as a buffer to hold the current output as was done in Example 1. (since this image will be discarded on the next read operation).
4. V3fwrite may be used to write out just the oldest index frame (index 0). This operation is different to that when V3fread is used to read a 3D image, in that case the whole 3D image is written.
5. The function section of the program is modeled after example thee. In this case the 3D image contains just 3 frames and we are computing the function corresponding to just the center frame (z index 1).
6. There is no need to buffer tm in the z direction
7. The first read reads n frames and the program loops until there are no more image in the file generating a result frame for each loop. Therefore, if the file has M image frames then only M – n + 1 image frames will be generated. For the Example 4 mean filter program the result will contain M-2 frames. In this way the operation of the buffer filter program differs from that of Example 3 the 3D program. The first and last frames (where computation overlaps the border of the 3D image) will not be generated; however, the remaining frames should contain the same values for each program.

Example 5 A program that processes multichannel or color images

```
/*********************************************************************//
* vcmean  Compute channel mean                                       */
/*********************************************************************/

#include "VisXV4.h"    /* VisionX structure include file       */
#include "Vutil.h"     /* VisionX utility header files         */

VXparam_t par[] =      /* command line structure               */
{
{    "if=",    0, " input file  vthresh: threshold images"},
{    "of=",    0, " output file "},
{     0,       0,    0}
};
#define  INIMAGE   par[0].val
#define  OUTIMAGE  par[1].val

int main(int argc, char** argv)
{
Vfstruct (im);
Vfstruct (om);
int x, y, c;
int thresh;
int chan, sum;

VXparse(&argc, &argv, par);  /* parse the command line         */

while ( Vfread( &im, INIMAGE) ) {
    if ( im.type != VX_PBYTE ) {
           fprintf (stderr, "error: image not byte type\n");
           exit (1);
    }
    Vfnewim (&om, VX_PBYTE, im.bbx, 1);
    for ( y = om.ylo; y <= om.yhi; y++) {
       for ( x = om.xlo; x <= om.xhi; x++) {
          sum = 0;
          for ( c = 0; c < im.chan; c++) {
               sum = sum + im.u[y][x * im.chan + c];
          }
          om.u[y][x] = sum / im.chan;
       }
    }
    Vfwrite( &im, OUTIMAGE);
  }
exit(0);
}
```

Multichannel images in Visionx are implemented by overloading the x-dimension of the image array structure. Multichannel images are identified by the x-range of the index being set to an exact multiple of the x-range in the bounding box.

A VisionX image may have any number of channels depending upon the application. Typical examples are three for color images and two for complex numbers.

Notes.

1. The program in example 5 will average the multi-channel values into a single channel image.
2. The program is limited to images of type unsigned byte.
3. The program loop limits are based on the output structure rather than the input structure
4. All parameters are the same for a multichannel image except for chan which is set to the number of channels and xlo and xhi, which are multiplied by the number of channels.

For a 3-channel image, imm, with xlo = 0 we have that:
imm.u[0][0] corresponds to the first channel of the first pixel
imm.u[0][1] corresponds to the second channel of the first pixel.
imm.u[0][3] corresponds to the first channel of the second pixel.