EE5470 Computer Vision Lab 2

Yuzhuo Sun NetId: ys684

Binary Image Processing

Lab 2 Questions

**1. Description of my program for image bounding (Section 4).**

When iterate through all image pixels from ylo to yhi and xlo to xhi, I classify each pixel into 3 kinds of pixels: 1, pixel has graylevel greater than 0; 2, pixel has graylevel equals 0 but its 4 neighbors contain graylevel greater than 0; 3, pixel has graylevel equals and all of its 4 neighbors’ graylevels equal to 0. As we defined above, the first type of pixels, I need to mark its graylevel to 128 as they are interior pixels; the second type of pixels, I classify them as the boarder pixel and mark its graylevel to 255 since they are adjacent to interior pixels; for the third type of pixels, I classify them as the general background therefore leave its graylevel to 0.

**2. Code for image bounding.**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* vtemp Compute local max operation on a single byte image \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "VisXV4.h" /\* VisionX structure include file \*/

#include "Vutil.h" /\* VisionX utility header files \*/

VXparam\_t par[] = /\* command line structure \*/

{ /\* prefix, value, description \*/

{ "if=", 0, " input file vtemp: local max filter "},

{ "of=", 0, " output file "},

{ 0, 0, 0} /\* list termination \*/

};

#define IVAL par[0].val

#define OVAL par[1].val

main(argc, argv)

int argc;

char \*argv[];

{

Vfstruct (im); /\* i/o image structure \*/

Vfstruct (tm); /\* temp image structure \*/

int y,x; /\* index counters \*/

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

Vfread(&im, IVAL); /\* read image file \*/

Vfembed(&tm, &im, 1,1,1,1); /\* image structure with border \*/

if ( im.type != VX\_PBYTE ) { /\* check image format \*/

fprintf(stderr, "vtemp: no byte image data in input file\n");

exit(-1);

}

for (y = im.ylo ; y <= im.yhi ; y++) { /\* compute the function \*/

for (x = im.xlo; x <= im.xhi; x++) { /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (im.u[y][x] > 0){

im.u[y][x] = 128;

}else if(im.u[y][x] == 0 && (tm.u[y - 1][x] > 0 || tm.u[y][x - 1] > 0 || tm.u[y][x + 1] > 0 || tm.u[y + 1][x] > 0)){

im.u[y][x] = 255;

}

else{

im.u[y][x] = 0;

}

}

}

Vfwrite(&im, OVAL); /\* write image file \*/

exit(0);

}

**3. Typescript copy for small image bounding.**

Input small Image:

0 1 2 3 4 5 6

5 0 0 0 0 0 0 0

4 0 1 1 0 0 0 0

3 0 0 1 0 0 0 0

2 0 0 0 0 3 3 0

1 0 0 0 3 3 3 0

0 0 0 0 0 0 0 8

Output small Image

0 1 2 3 4 5 6

5 0 255 255 0 0 0 0

4 255 128 128 255 0 0 0

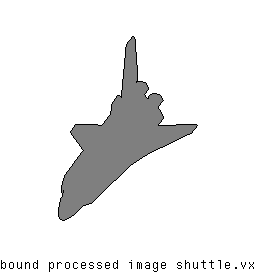
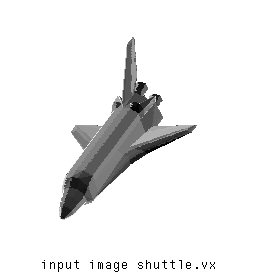
3 0 255 128 255 255 255 0

2 0 0 255 255 128 128 255

1 0 0 255 128 128 128 255

0 0 0 0 255 255 255 128

**4. Full size image bounding**



**5. Description of image labeling (Section 5)**

For this part, I used DFS (Depth First Search) procedure. When iterate through all pixels in a given image. Once we encounter a non-background pixel (with graylevel above 0), I will call a separate function to change its value to a label value starts from 1. Then I will use the copy of the original image as a reference to mark its value to 0. And call label function to its 4 neighbors and so on, until the recursive call encounters a background value. Then return to iteration once again and increase the label value until we iterate through all given image pixels.

**6. Code for image labeling**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* vtemp Compute local max operation on a single byte image \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "VisXV4.h" /\* VisionX structure include file \*/

#include "Vutil.h" /\* VisionX utility header files \*/

VXparam\_t par[] = /\* command line structure \*/

{ /\* prefix, value, description \*/

{ "if=", 0, " input file vtemp: local max filter "},

{ "of=", 0, " output file "},

{ 0, 0, 0} /\* list termination \*/

};

#define IVAL par[0].val

#define OVAL par[1].val

void mark\_region(int, int, int);

Vfstruct (im); /\* i/o image structure \*/

Vfstruct (tm); /\* temp image structure \*/

main(argc, argv)

int argc;

char \*argv[];

{

int y,x; /\* index counters \*/

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

Vfread(&im, IVAL); /\* read image file \*/

Vfembed(&tm, &im, 1,1,1,1); /\* image structure with border \*/

int i = 1; /\* region sequence \*/

if ( im.type != VX\_PBYTE ) { /\* check image format \*/

fprintf(stderr, "vtemp: no byte image data in input file\n");

exit(-1);

}

for (y = im.ylo ; y <= im.yhi ; y++) {

for (x = im.xlo; x <= im.xhi; x++) {

if(tm.u[y][x] == 0) {continue; }

mark\_region(x, y, i);

i++;

}

}

Vfwrite(&im, OVAL); /\* write image file \*/

exit(0);

}

/\* function to mark the region \*/

void mark\_region(int x, int y, int cnt)

{

if (tm.u[y][x] == 0) {return; }

im.u[y][x] = cnt;

tm.u[y][x] = 0; /\* tag the visited pixel \*/

mark\_region(x + 1, y, cnt);

mark\_region(x - 1, y, cnt);

mark\_region(x, y + 1, cnt);

mark\_region(x, y - 1, cnt);

}

**7. Typescript copy for small image labeling**

Input image

0 1 2 3 4 5 6 7 8 9

9 0 0 0 0 0 0 0 0 0 0

8 0 255 255 255 255 0 0 0 0 0

7 0 0 0 0 255 0 0 0 0 0

6 0 255 255 0 255 255 0 0 255 0

5 0 255 255 0 255 255 0 0 255 0

4 0 255 255 0 0 0 0 0 255 0

3 0 255 255 0 0 255 255 255 255 0

2 0 0 0 0 255 255 0 0 0 0

1 0 0 0 255 0 0 255 255 255 0

0 0 0 0 0 0 0 255 255 255 0

Output image

0 1 2 3 4 5 6 7 8 9

9 0 0 0 0 0 0 0 0 0 0

8 0 5 5 5 5 0 0 0 0 0

7 0 0 0 0 5 0 0 0 0 0

6 0 4 4 0 5 5 0 0 3 0

5 0 4 4 0 5 5 0 0 3 0

4 0 4 4 0 0 0 0 0 3 0

3 0 4 4 0 0 3 3 3 3 0

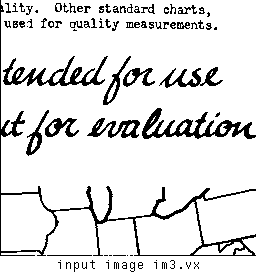
2 0 0 0 0 3 3 0 0 0 0

1 0 0 0 2 0 0 1 1 1 0

0 0 0 0 0 0 0 1 1 1 0

**8. Full size image labeling**

For output image, I used vx tf=31 operation to multiply each pixel value by 31 times to make it more easy to spot different regions (Since the original difference is 1 in my program, it is hard to spot difference between adjacent regions). Then I used vx bf=30 operation to offset the background to white since the previous multiply operation will change the background to grey.ß

****