**Biological Cybernetics Lab. Training**

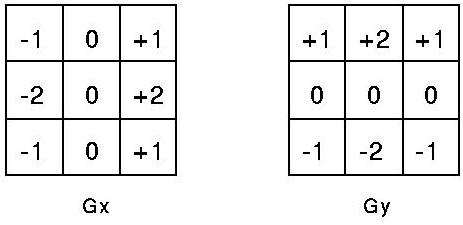
Section 1. Sobel operator

The Sobel operator is used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation which it produces is relatively crude, in particular for high frequency variations in the image.

In simple terms, the operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or "smoothly" the image changes at that point, and therefore how likely it is that that part of the image represents an edge, as well as how that edge is likely to be oriented. In practice, the magnitude (likelihood of an edge) calculation is more reliable and easier to interpret than the direction calculation.

Mathematically, the gradient of a two-variable function (here the image intensity function) is at each image point a 2D vector with the components given by the derivatives in the horizontal and vertical directions. At each image point, the gradient vector points in the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction. This implies that the result of the Sobel operator at an image point which is in a region of constant image intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values.

Mathematically, the operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations, the computations are as follows:

****

For a gray-colored image, the image can be defined as a m x n matrix (or m x n x 3) (their RGB values are the same). Then one pixel value can be represented as I(x,y) at location (x,y). For edge detection, we apply the above Sobel operator to a set of pixels around I(x,y).

*Gx = - 1 \* I(x-1, y-1) + 0\* I(x,y-1) + 1 \* I(x+1,y-1)*

*- 2 \* I(x-1,y) + 0\*I(x,y) + 2 \* I(x+1,y)*

*-1 \* I(x-1, y-1) + 0\*I(x,y+1) + 1 \* I(x+1,y+1)*

*Gy = + 1 \* I(x-1, y-1) + 2\* I(x,y-1) + 1 \* I(x+1,y-1)*

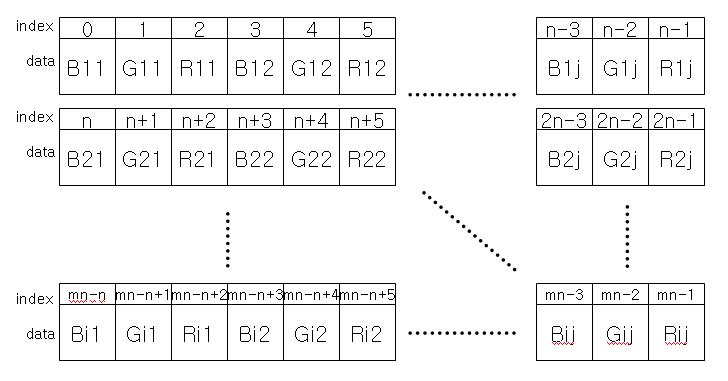
*0 \* I(x-1,y) + 0\*I(x,y) + 0 \* I(x+1,y)*

*-1 \* I(x-1, y-1) - 2\*I(x,y+1) - 1 \* I(x+1,y+1)*

The magnitude of the gradient is then calculated using the following formula. Each pixel value will be assigned with this magnitude.

grad_mag

Here, we use the targa (tga) file format for image processing. The tga format describes one image as one-dimensional character string. One image pixel consists of BGR (blue, green, red) color values where each color has a value from 0 to 255. One character byte can be assigned for each BGR color. Thus, one image pixel can be described with 3 bytes of character bytes. If an image file has a size of 200 x 100 (width x height), the total number of bytes is 200 x 100 x 3 bytes. If there is an array of character bytes whose size is 60000, for example, char temp[60000]; , then the temp array can store the image. The following picture shows how the whole image is stored.

****

For this section, we use dynamic memory allocation with *malloc* function. This is just example code. If you want to insert this code, you can insert this code in your mind.

#include <stdio.h>

int main()

{

FILE \*out, \*in;

unsigned char \*buffer, \*buffout;

short targaheader[9]={ 0,2,0,0,0,0,111,222,8216 };

unsigned char sname[100];

unsigned char temp;

int width, height, i, j, k, size;

in=fopen("original.tga", "rb");

// read first the header component

fread(targaheader, sizeof(targaheader), 1, in);

width = targaheader[6]; // read picture width

height = targaheader[7]; // read picture height

size=width\*height\*3;

// allocate memory for buffer variable

**buffer** = (unsigned char \*) malloc(width\*height\*3);

// save the image into buffer

fread(**buffer**, width \* height \*3 , 1, in);

fclose(in);

**buffout** = (unsigned char \*) malloc(width\*height\*3);

// copy buffer to buffout

for(i=0; i <= width\*height\*3; i++) buffout[i] = buffer[i];

out=fopen("test.tga", "wb");

fwrite(targaheader, sizeof(targaheader), 1, out);

fwrite(**buffout**, width \* height \*3 , 1, out);

fclose(out); free(buffer); free(buffout);

}

The above program simply copies one image file with *tga* format into another file called ‘*test.tga’*.

(1) Now you need to design a program for edge detection using the Sobel operator. Initially you need to change the colored image into a gray-colored image. If BGR values for one pixel are all the same, then you can have a gray color for the pixel. Thus, first calculate the average over the BGR values, that is, 3 bytes of pixel values and then set the average value into the BGR values. For example, R=120, G=240, B=45, then the average value (120+240+45)/3 = 135 will be re-assigned into R=135, G=135, B=135. Then you get a gray-colored pixel. You need to change all the pixel values in this way. Then you save the image buffer into a file. Then you get a gray-colored image file.

(2) Apply the Sobel operator equation shown above to each gray-colored pixel. Calculate the magnitude of the gradient for each pixel, and then save the image buffer into a file. You can see the result of edge detection in the image file. The result for the above example would look like

|  |  |
| --- | --- |
| temp | temp2 |