

Homework 2

Convolution and Edge Detection Using the Sobel Operator

Eric Liskay

ECE 590
Spring 2012

Objective

The objective of this homework is to Sobel edge detection and how it can be implemented in VHDL. The ultimate goal is to use this in my final project, the SIMD Image Processor.

Algorithm

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$
$$G = \sqrt{G_x^2 + G_y^2}$$

The above figure¹ shows how Sobel edge detection is done mathematically. The Sobel operator uses two 3x3 kernels which are convolved with a 3x3 array of pixels from the image, denoted as A . G_x and G_y are the derivatives of the convolution. G_x is for horizontal changes and G_y is for vertical changes. The $*$ denotes the 2-dimensional convolution operation.

The output G , the gradient magnitude, is obtained by squaring G_x and G_y , adding them, and then taking the square root of the resulting sum. The approximate magnitude can also be computed by just adding the absolute values of G_x and G_y . This may be less accurate, but is faster to compute.

¹ http://en.wikipedia.org/wiki/Sobel_operator

Implementation in VHDL

In my design, I will compute the derivatives, G_x and G_y , by multiplying the values at the corresponding indices in each matrix and adding the results. The VHDL code for this can be seen below. Since I am working with color images, I will simplify operations by only looking at the red channel. inputA is an array of pixels with x and y representing the indices and 0 to select the red channel.

```
variable Gx, Gy : signed(15 downto 0) := (others=> '0');
variable Gu : unsigned(15 downto 0);
variable Gsqd : unsigned(31 downto 0);

constant Sobelx : kernel := ((-1,0,1),(-1,0,2),(-1,0,1));
constant Sobely : kernel := ((-1,-2,-1),(0,0,0),(1,2,1));

Gx := (others=>'0');
Gy := (others=>'0');
for y in 0 to 2 loop
    for x in 0 to 2 loop
        Gx := Gx + conv_signed((conv_integer(unsigned(inputA(x,y)(0))) * Sobelx(x,y)),16);
        Gy := Gy + conv_signed((conv_integer(unsigned(inputA(x,y)(0))) * Sobely(x,y)),16);
    end loop;
end loop;
```

I used a temporary variable to hold the output of the sums of G_x^2 and G_y^2 .

```
Gsqd := conv_unsigned((conv_integer(Gx)**2) + (conv_integer(Gy)**2),32);
```

I then use the function² below to calculate the square root to find G.

```
Gu := sqrt(Gsqd);
```

```
function sqrt ( d : UNSIGNED ) return UNSIGNED is
    variable a : unsigned(31 downto 0):=d;  --original input.
    variable q : unsigned(15 downto 0):=(others => '0');  --result.
    variable left,right,r : unsigned(17 downto 0):=(others => '0');
    variable i : integer:=0;
begin
    for i in 0 to 15 loop
        right(0):='1';
        right(1):=r(17);
        right(17 downto 2):=q;
        left(1 downto 0):=a(31 downto 30);
        left(17 downto 2):=r(15 downto 0);
        a(31 downto 2):=a(29 downto 0);  --shifting by 2 bit.
        if ( r(17) = '1') then
            r := left + right;
        else
            r := left - right;
        end if;
        q(15 downto 1) := q(14 downto 0);
        q(0) := not r(17);
    end loop;
    return q;
end sqrt;
```

Once I have the magnitude, G, I then have to cap the maximum value that can be held in 8-bits, the size of the channel. I do this by assigning all 1's to the magnitude if it is currently larger than 255. I then assign the lower 8 bits to the center pixel of the output image. Each channel is assigned the same value which will make the image display in grayscale.

```
if(Gu > conv_UNSIGNED(255, 16)) then
    Gu := (others => '1');
end if;

for c in 0 to (IMAGE_CHANNELS-1) loop
    outputC(1,1)(c) <= conv_std_logic_vector(Gu,8);
end loop;
```

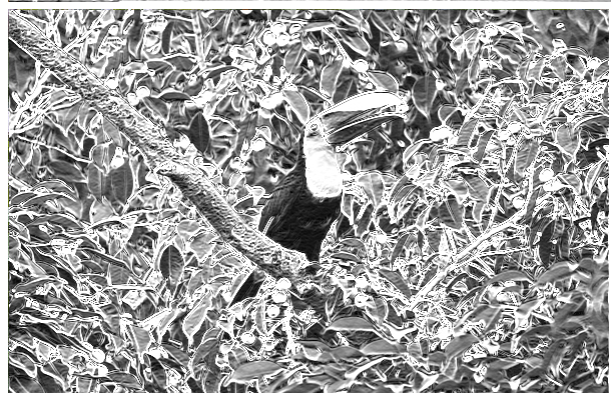
² <http://vhdlguru.blogspot.com/2010/03/vhdl-function-for-finding-square-root.html>

Results(Images with edges highlighted by Sobel)

Original mage



Sobel Processed Image



Original Image



Sobel Processed Image



Original Image



Sobel Processed Image



Original Image



Sobel Processed Image

