Lab B
Sobel Filter

R09921067 朱祐葳

Outline

- Sobel Filter Introduction
- Implementation
- Evaluation

Outline

- Sobel Filter Introduction
- Implementation
- Evaluation

Sobel Filter

- Sobel and Feldman presented the idea of an "Isotropic 3x3 Image Gradient Operator" at a talk at SAIL in 1968
- Used in image processing and computer vision
- Edge detection algorithms where it creates an image emphasising edges

Isotropic 3x3 Image Gradient Operator

Irwin Sobel

Sobel Filter

The neighbors group into antipodal pairs: (a,i) (b,h) (c,g) (f,d).

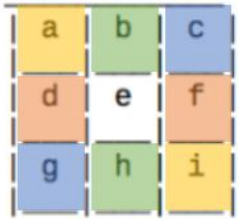
Thus for a point on a Cartesian grid and its eight neighbors having density values as shown

a	b	c
d	e	f
g	h	i

Sobel Filter

The neighbors group into antipodal pairs: (a,i) (b,h) (c,g) (f,d).

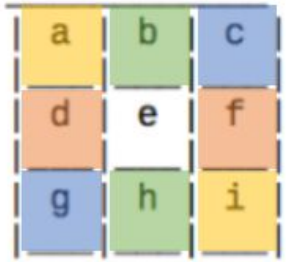
Thus for a point on a Cartesian grid and its eight neighbors having density values as shown



Sobel Filter

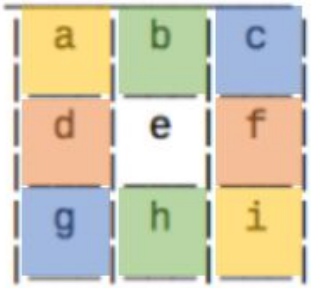
Define the magnitude of the directional derivative estimate vector 'g' for a given neighbor as

|g| = <**density difference**>/<**distance to neighbor**>



Sobel Filter

We can derive G



$$\Rightarrow G = (c-g)/4 * [1, 1] + (a-i)/4 * [-1, 1] + (b-h)/2 * [0, 1] + (f-d)/2 * [1, 0]$$

$$\Rightarrow G = [(c-g-a+i)/4 + (f-d)/2, (c-g+a-i)/4 + (b-h)/2]$$

$$\Rightarrow G' = 4*G = [c-g-a+i + 2*(f-d), c-g+a-i + 2*(b-h)]$$

Sobel Filter

We can derive G and obtain the following matrix

$$G' = 4*G = [c-g-a+i + 2*(f-d),\ c-g+a-i + 2*(b-h)]$$

It is useful to express this as weighted density summations using the following weighting functions for x and y components:

-1	0	1
-2	0	2
-1	0	1

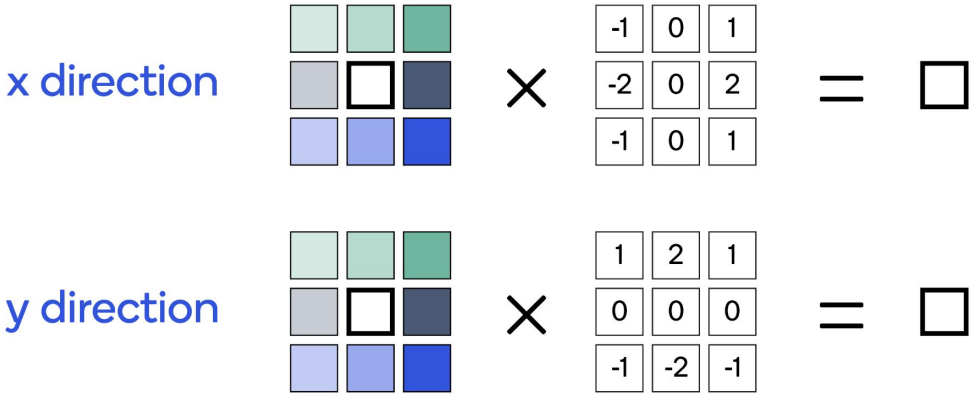
x-component

1	2	1
0	0	0
-1	-2	-1

y-component

Sobel Filter

We can derive G and obtain the following matrix



Sobel Filter

The resulting gradient approximations can be combined to give the gradient magnitude

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

To simplify, we have

$$G = |G_x| + |G_y|$$

Outline

- Sobel Filter Introduction
- Implementation
- Evaluation

Sobel Filter Operation

```
const char x_op[3][3] = {
                {-1,0,1},
                {-2,0,2},
                {-1,0,1}};
```

X and Y filter

```
const char y_op[3][3] = {
                {1,2,1},
                {0,0,0},
                {-1,-2,-1}};

//Compute approximation of the gradients in the X-Y direction

for(i=0; i < 3; i++){
  for(j = 0; j < 3; j++){
#pragma HLS unroll factor=3

    // X direction gradient
    x_weight = x_weight + (window->getval(i,j) * x_op[i][j]);

    // Y direction gradient
    y_weight = y_weight + (window->getval(i,j) * y_op[i][j]);
  }
}
```

Calculation

Sobel Filter Operation

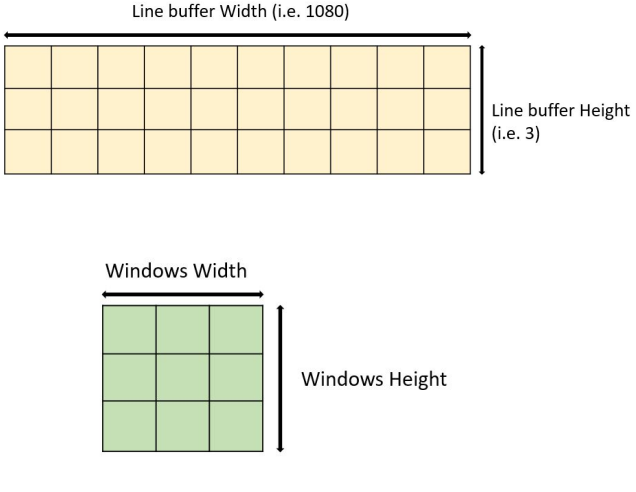
```
edge_weight = ABS(x_weight) + ABS(y_weight);

edge_val = (255-(unsigned char)(edge_weight));

//Edge thresholding
if(edge_val > 200)
  edge_val = 255;
else if(edge_val < 100)
  edge_val = 0;

pixel.R = pixel.G = pixel.B = edge_val;

return pixel;
```

Gradient

Threshold

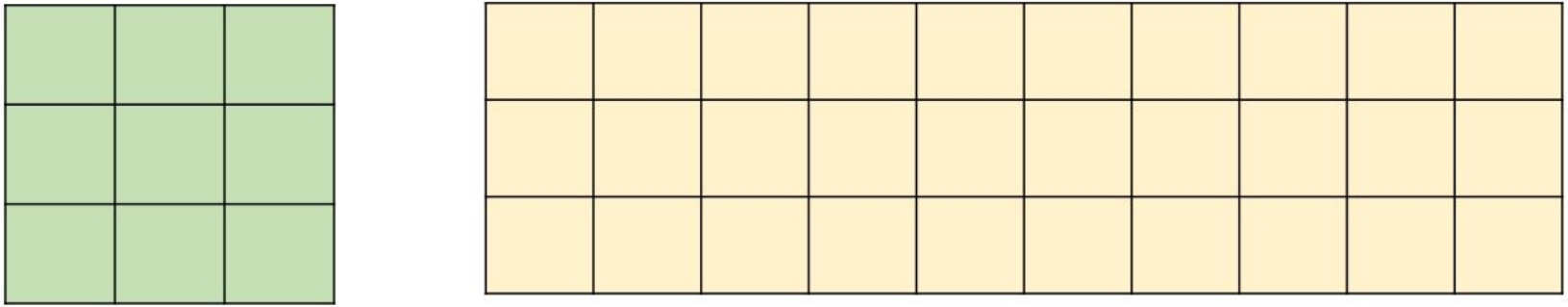
Sobel Filter

Line Buffer



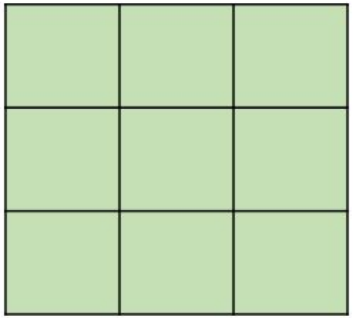
Sobel Filter

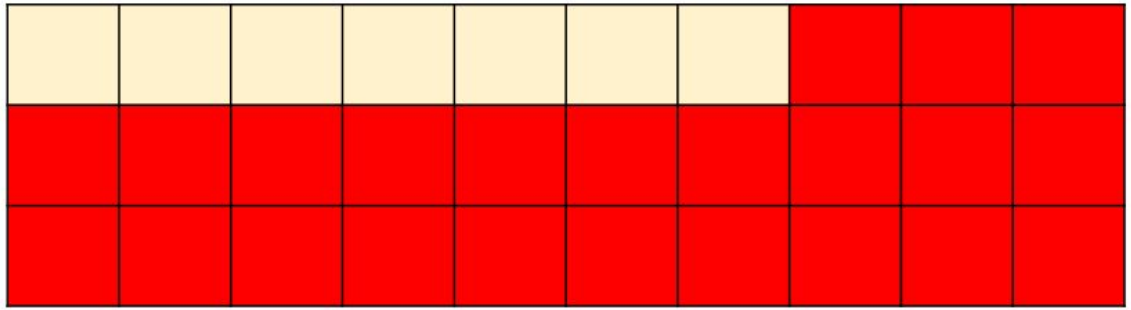
Line Buffer



Sobel Filter

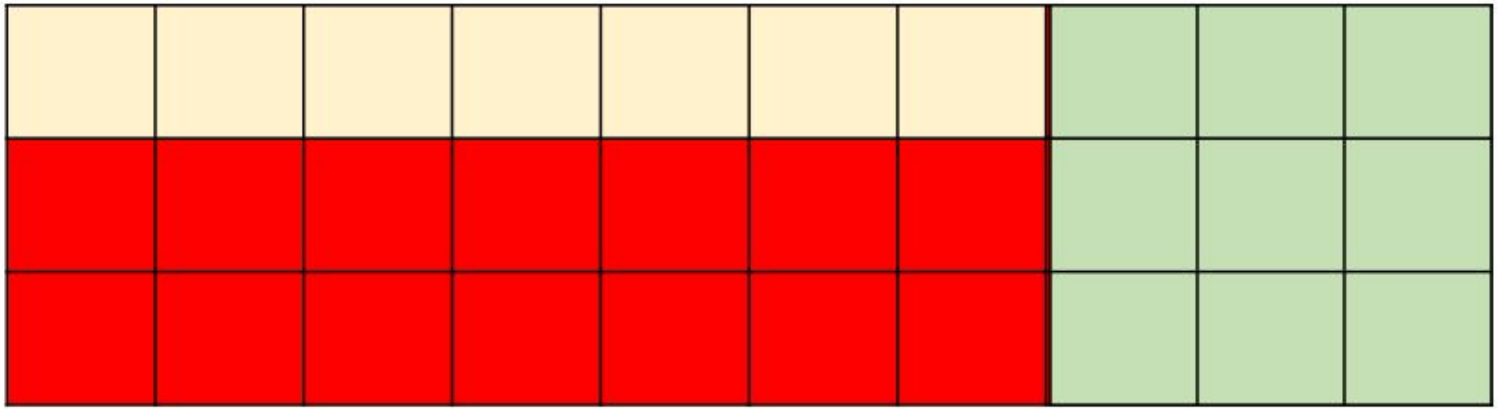
Line Buffer





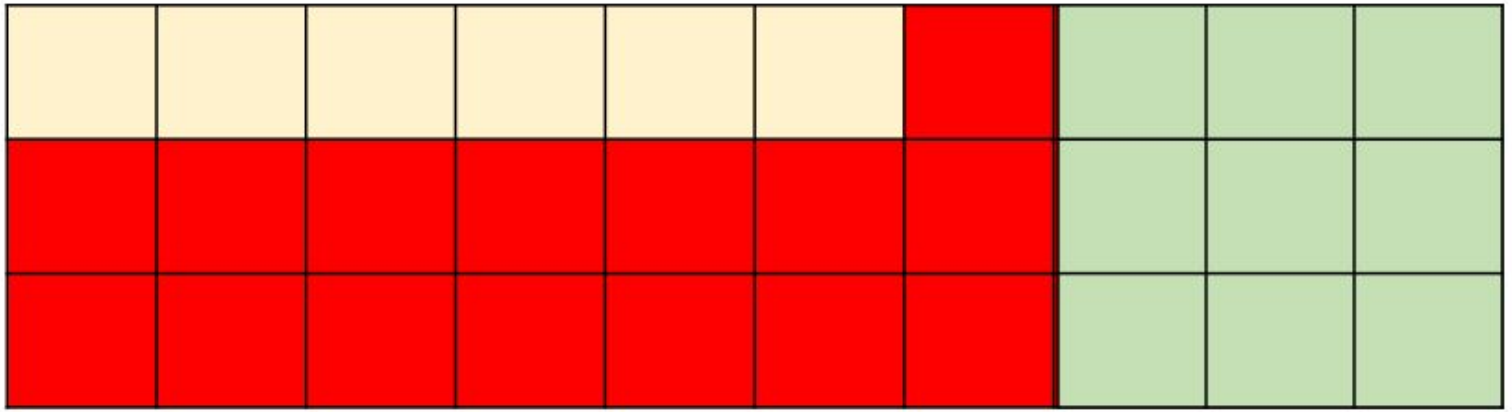
Sobel Filter

Line Buffer



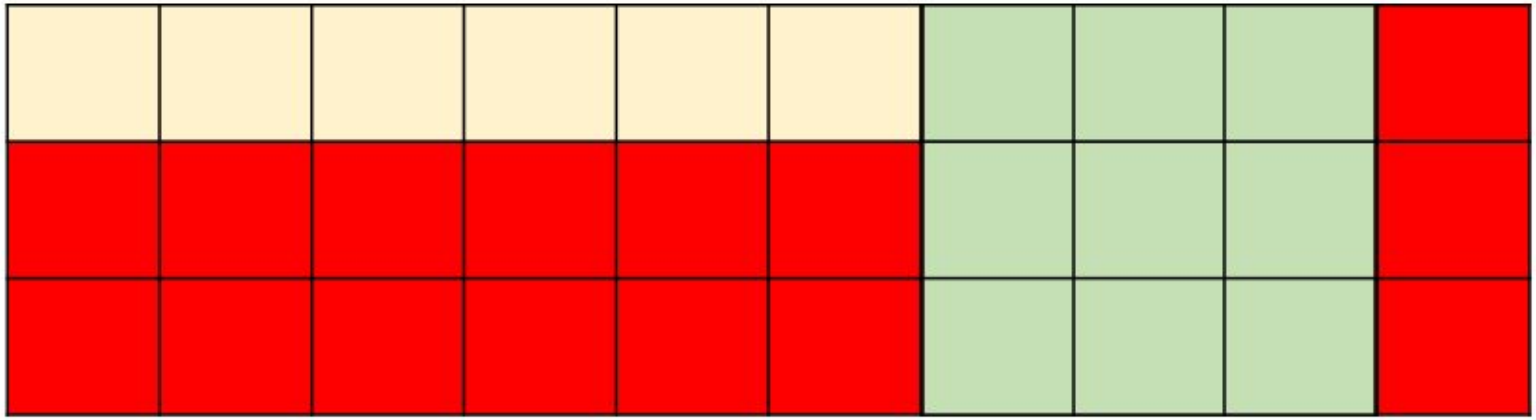
Sobel Filter

Line Buffer - Read



Sobel Filter

Line Buffer - Slide



Sobel Filter Main

```
for(row = 0; row < ROWS+1; row++){
  for(col = 0; col < COLS+1; col++){

    //Line Buffer fill
    if(col < cols){
  buff_A.shift_up(col);
  temp = buff_A.getval(0,col);
    }

    //There is an offset to accommodate the active pixel region
    //There are only MAX_WIDTH and MAX_HEIGHT valid pixels in the image
    // Insert new pixel into linebuffer
    if(col < cols & row < rows){
  RGB new_pix;
  AXI_PIXEL input_pixel;
  input_pixel = inter_pix[row][col];
  new_pix.B = input_pixel.data.range(7,0);
  new_pix.G = input_pixel.data.range(15,8);
  new_pix.R = input_pixel.data.range(23,16);
  tempx = new_pix;;
  buff_A.insert_bottom(rgb2y(tempx),col);
    }

    //Shift the processing window to make room for the new column
    buff_C.shift_right();

    //The Sobel processing window only needs to store luminance values
    //rgb2y function computes the luminance from the color pixel
    if(col < cols){
  buff_C.insert(buff_A.getval(2,col),0,2);
  buff_C.insert(temp,1,2);
  buff_C.insert(rgb2y(tempx),2,2);
    }
```

buff_A : Linebuffer
buff_C : Windows

Insert Linebuffer

Insert Windows

Outline

- Sobel Filter Introduction
- Implementation
- Evaluation

Image

- Input - Image (1920 x 1080)

  Output - Image (1920 x 1080)

Pragma

- pragma HLS pipeline
  - Initiation interval II = 1
- pragma HLS loop_flatten
  - Allows nested loops to be flattened into a single loop hierarchy with improved latency.
- pragma HLS dependence
  - Provide additional information that can overcome loop-carry dependencies and allow loops to be pipelined (or pipelined with lower intervals).

HLS Dependence

```
#pragma HLS dependence variable=&buff_A false

      if(col < cols){
    buff_A.shift_up(col);
    temp = buff_A.getval(0,col);
      }
```

```
template <typename T, int LROW, int LCOL>
  void ap_linebuffer<T,LROW,LCOL>::shift_up(int col)
{
  int i;
  for(i = LROW-1; i > 0; i--){
    M[i][col] = M[i-1][col];
  }
}
```

HLS Dependence

```
#pragma HLS dependence variable=&buff_A false

      if(col < cols){
    buff_A.shift_up(col);
     temp = buff_A.getval(0,col);
      }
```

```
template <typename T, int LROW, int LCOL>
  void ap_linebuffer<T,LROW,LCOL>::shift_up(int col)
{
  int i;
  for(i = LROW-1; i > 0; i--){
    M[i][col] = M[i-1][col];
  }
}
```

HLS Dependence

```
#pragma HLS dependence variable=&buff_A false

      if(col < cols){
    buff_A.shift_up(col);
     temp = buff_A.getval(0,col);
      }
```

```
template <typename T, int LROW, int LCOL>
  void ap_linebuffer<T,LROW,LCOL>::shift_up(int col)
{
  int i;
  for(i = LROW-1; i > 0; i--){
    M[i][col] = M[i-1][col];
  }
}
```

HLS Dependence

```
#pragma HLS dependence variable=&buff_A false

      if(col < cols){
    buff_A.shift_up(col);
     temp = buff_A.getval(0,col);
      }
```

False Dependence!

```
template <typename T, int LROW, int LCOL>
  void ap_linebuffer<T,LROW,LCOL>::shift_up(int col)
{
  int i;
  for(i = LROW-1; i > 0; i--){
    M[i][col] = M[i-1][col];
  }
}
```

HLS Unroll

```
  for(i=0; i < 3; i++){
#pragma HLS unroll factor=3
    for(j = 0; j < 3; j++){
#pragma HLS unroll factor=3

      // X direction gradient
      x_weight = x_weight + (window->getval(i,j) * x_op[i][j]);

      // Y direction gradient
      y_weight = y_weight + (window->getval(i,j) * y_op[i][j]);
    }
  }
```

Latency

Cosimulation Report for 'sobel_filter'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	2088493	2088493	2088493	NA	NA	NA

Export the report(.html) using the Export Wizard

Cosimulation Report for 'sobel_filter'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	2076616	2076616	2076616	NA	NA	NA

Export the report(.html) using the Export Wizard

With Loop_flaten off

Latency

Pragma Type	Latency (Absolute Max)
None	0.8 (sec)
Pragma Unroll	0.156 (sec)
Manual Unroll	0.156 (sec)
Pipeline	10.383 (ms)
Pipeline & Loop Flatten	10.448 (ms)
Pipeline & Loop Flatten & Unroll	10.442 (ms)

Latency

Pragma Type	Latency (Absolute Max)	
None	0.8 (sec)	
Pragma Unroll	0.156 (sec)	
Manual Unroll	0.156 (sec)	5.12X
Pipeline	10.383 (ms)	77.0X
Pipeline & Loop Flatten	10.448 (ms)	
Pipeline & Loop Flatten & Unroll	10.442 (ms)	

Utilization

Utilization

Pragma Type	DSP48E	FF	LUT
None	4	587	1104
Pragma Unroll	2	516	981
Manual Unroll	2	523	985
Pipeline	2	1191	1332
Pipeline & Loop Flatten	2	1112	1181
Pipeline & Loop Flatten & Unroll	2	1070	1176

Utilization

Pragma Type	DSP48E	FF	LUT
None	4	587	1104
Pragma Unroll	2	516	981
Manual Unroll	2	523	985
Pipeline	2	1191	1332
Pipeline & Loop Flatten	2	1112	1181
Pipeline & Loop Flatten & Unroll	2	1070	1176

Utilization

Pragma Type	DSP48E	FF	LUT
None	4	587	1104
Pragma Unroll	2	516	981
Manual Unroll	2	523	985
Pipeline	2	1191	1332
Pipeline & Loop Flatten	2	1112	1181
Pipeline & Loop Flatten & Unroll	2	1070	1176

Utilization

Pragma Type	DSP48E	FF	LUT
None	4	587	1104
Pragma Unroll	2	516	981
Manual Unroll	2	523	985
Pipeline	2	1191	1332
Pipeline & Loop Flatten	2	1112	1181
Pipeline & Loop Flatten & Unroll	2	1070	1176

Conclusion

- Trade off between latency and utilization
- Dependence Pragma do nothing (Compiler??)
- Linebuffer data structure

Sobel Filter Result

Original

Sobel Filter

Sobel Filter Result

Threshold = 50

Threshold = 100

Sobel Filter Result

Original

Threshold = 100

Threshold = 250

Sobel Filter

We can derive G and obtain the following matrix

$$G' = 4*G = [c-g-a+i + 2*(f-d),\ c-g+a-i + 2*(b-h)]$$

It is useful to express this as weighted density summations using the following weighting functions for x and y components:

-1	0	1
-2	0	2
-1	0	1

x-component

1	2	1
0	0	0
-1	-2	-1

y-component

Sobel Filter Result

Weight = 1

Weight = 2

Question

1. What is the function of #pragma loop_flatten false
2. What kind of data structure is used in Sobel Filter

Sobel Filter

Github Link : HLS_LabB_SobelFilter

Thank you!