Edge Detection

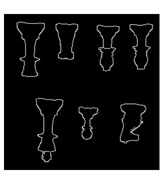
- The purpose of Edge Detection is to find jumps in the brightness function (of an image) and mark them.

1

Consider this picture

2

We would like its output to be



3

Edge Detection

- So, to repeat: The purpose of Edge Detection is to find jumps in the brightness function (of an image) and mark them.

Before we get into details, we need to detour and introduce the concept of Convolution.

4

Introducing Convolution

- Is an operation between two tables of numbers, usually between an image and weights.
- Typically, if one table is smaller, it is on the right of the operator

$\circledast$

$$\underset{\text{Image}}{\begin{bmatrix} -1 & +1 \\ -2 & +2 \end{bmatrix}} \circledast \underset{\text{Weights}}{\begin{bmatrix} 0 & 2 \\ 1 & 3 \end{bmatrix}} = ???$$

5

Executing a Convolution

- Pad left array with several zeros.
- Do a double-flip or diagonal flip on right array.
- Then compute the weighted sum.
- (In practice we don’t do double flip.)

6

Convolution: Step One
Padding an array with zeros

$$\begin{bmatrix} -1 & +1 \\ -2 & +2 \end{bmatrix}$$

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

Do we start Convolution at the start of original numbers or from the padded numbers?

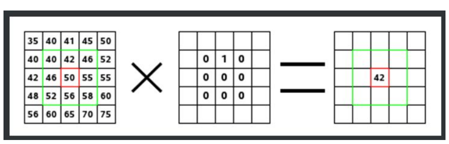
7

Convolution : Step Two
Double Flip the second array

$$\begin{bmatrix} 0 & 2 \\ 1 & 3 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}$$

8

Convolution: Step Three
Computing Weighted Sum



The sum of the convolution goes in the upper left hand corner only for 2x2 tables. Any larger tables should have an odd number of rows and columns and the sum will go in the center.

(40*0)+(42*1)+(46*0) + (46*0)+(50*0)+(55*0) + (52*0)+(56*0)+(58*0) = 42.

9

Computing Weighted Sum

The computed sum goes in this location, but not on the original image, instead, it goes on an output table.

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

$$\begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}$$

The weights

3*0 + 1*0 + 2*0 + 0*0

This calculates the weighted sum.

Summing these numbers after weighting them by the individual weights

[0]

•Note: each weighted sum results in one number.
•This number gets placed in the output array in the position that the inputs came from.

10

The previous slide….

- Was an example of a one-location convolution.
- If we move the location of the numbers being summed, we have scanning convolution.
- The next slides show the scanning convolution.

11

Computing Weighted Sum

First compute all columns in first row, then move on to the second row, and so on.

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

$$\begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}$$

The weights

3*0 + 1*0 + 2*0 + 0*0

This calculates the next weighted sum.

Summing these numbers

•This number gets placed in the output array in the NEXT position.

[0 0]

12

Computing Weighted Sum

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

$$\begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}$$

The weights

3*2 + 1*0 + 2*0 + 0*0

This calculates the next weighted sum.

Summing these numbers

- This number gets placed in the output array in the NEXT position.

$$\begin{bmatrix} 0 & 0 & \\ & & 6 \end{bmatrix}$$

13

Computing Weighted Sum

- In example, have two arrays of four numbers each
- One is the image, the other is the weights.
- The result is the weighted sums (the output)

$$\underbrace{\begin{bmatrix} -1 & +1 \\ -2 & +2 \end{bmatrix}}_{\text{(image)}} \circledast \underbrace{\begin{bmatrix} 0 & 2 \\ 1 & 3 \end{bmatrix}}_{\text{(weights)}} = \underbrace{\begin{bmatrix} 0 & -2 & +2 \\ -1 & -6 & +7 \\ -2 & -4 & +6 \end{bmatrix}}_{\text{(weighted sums)}}$$

14

Edge Detection

Now that we have defined convolution, and know how to execute it, let us put aside the concept of Convolution, while we consider a simple approach to detecting a steep jump in brightness values in a row.
(After that, we will employ the notion of convolution).

15

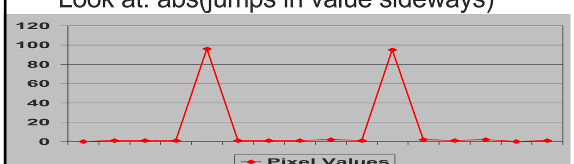
Consider this picture

16

Consider a row of values in picture

1 2 1 0 98 99 98 97 99 98 1 2 1 2

Look at: abs(jumps in value sideways)



17

Edge Detection

- Create the algorithm in pseudocode:

  while row not ended // keep scanning until end of row

  select the next A and B pair, which are neighboring pixels.

  diff = B – A //formula to show math

  if abs(diff) > Threshold //(THR)

  mark as edge

Above is a simple solution to detecting the differences in pixel values that are side by side.

18

Edge Detection: One-location Convolution

- diff = B – A is the same as:

A	B	⊛	-1	+1

Box 1   Box 2 (“weights”)

Convolution symbol

Place box 2 on top of box 1, multiply.
-1 * A and +1 * B
Result is –A + B which is the same as
B – A

19

Edge Detection

- Create the algorithm in pseudocode:

```
while row not ended          // keep scanning until end of row
    select the next A and B pair
    diff = [A][B] ⊛ [-1][+1]          //formula to show math
        if abs(diff) > Threshold      //(THR)
            mark as edge
```

Above is a simple solution to detecting the differences in pixel values that are side by side.

20

Edge Detection

- Create the algorithm in pseudocode:

```
while row not ended          // keep scanning until end of row
    select the next A and B pair
    diff = [A][B] ⊛ [-1][+1]          //formula to show math
        if abs(diff) > Threshold      //(THR)
            mark as edge
```

Note that in this algorithm, we are actually doing a scanning convolution, the scan is hidden in the while loop

21

Edge Detection: Pixel Values Become Gradient Values

- 2 pixel values are derived from two measurements
  - Horizontal

A	B	$\circledast$	-1	+1

  - Vertical

A	$\circledast$	-1
B		+1

Note: A,B pixel pair will be moved over whole image to get different answers at different positions on the image

22

The Resulting Vectors

- Two values are then considered vectors
- The vector is a pair of numbers:
  - [Horizontal answer, Vertical answer]
- This pair of numbers can also be represented by magnitude and direction

23

Edge Detection: Vectors

- The magnitude of a vector is the square root of the numbers from the convolution

$$\sqrt{(a)^2 + (b)^2}$$

The answer to this equation yields the difference of brightness between neighboring pixels. Higher number means a greater sudden change in brightness.

a is horizontal answer
b is vertical answer

We can plot the output of this equation onto a 2 dimensional image to show edges.

24

Edge Detection: Deriving Gradient, the Math

- The gradient is made up of two quantities:
  - The derivative of I with respect to x
  - The derivative of I with respect to y

∂ = derivative symbol

$\triangleq$ = is defined as

$$\vec{\nabla} \vec{I} \triangleq \left[\frac{\partial I}{\partial x} , \frac{\partial I}{\partial y} \right]$$

$$\| \vec{\nabla} I \|$$

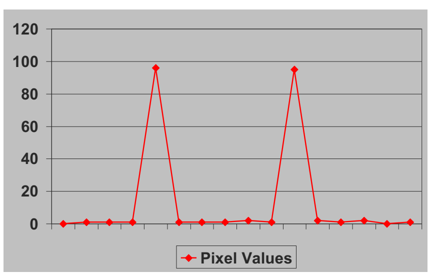
So the gradient of I is the magnitude of the gradient. Arrived at mathematically by the “simple” Roberts algorithm.

$$\sqrt{\left(\frac{\partial I}{\partial x} \right)^2 + \left(\frac{\partial I}{\partial y} \right)^2}$$

The Sobel method takes the average Of 4 pixels to smooth (applying a smoothing feature before finding edges.

25

Showing abs(diff B-A)

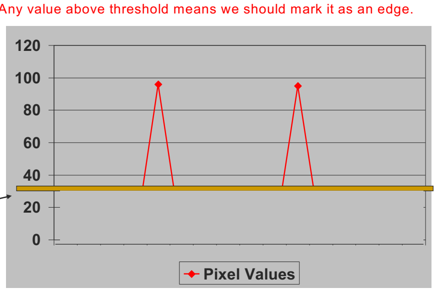


26

Effect of Thresholding

Any value above threshold means we should mark it as an edge.

Threshold Bar



27

Thresholding the Gradient Magnitude

- Whatever the gradient magnitude is, for example, in the previous slide, with two blips, we picked a threshold number to decide if a pixel is to be labeled an edge or not.
- The next three slides will shows one example of different thresholding limits.

28

Gradient Magnitude Output

This image takes the original pixels from chess.pgm, and replaces the each pixel's value using the magnitude formula discussed earlier.

This image does not use any threshold.

29

Consider this picture

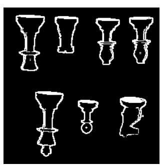
30

Consider this picture

31

Magnitude Output with a low bar (threshold number)

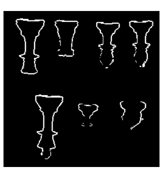
This image takes the pixel values from the previous image and uses a threshold to decide whether it is an edge and then for edges it replaces the pixel's value with a 255. If it is not an edge, it replaces the pixel's value with a 0.



32

Magnitude Output with a high bar (threshold number)

The higher threshold means that greater changes in brightness must be present to be considered an edge.



Edges have thinned out, but horses head and other parts of the pawn have disappeared. We can hardly see the edges on the bottom two pieces.

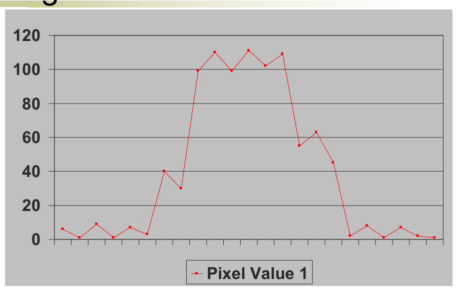
33

Magnitude Formula in the c Code

```
/* Applying the Magnitude formula in the code*/
      maxival = 0;
      for (i=mr;i<256-mr;i++)
      { for (j=mr;j<256-mr;j++)
         {
           ival[i][j]=sqrt((double)((outpicx[i][j]*outpicx[i][j])
   +
                             (outpicy[i][j]*outpicy[i][j]));
           if (ival[i][j] > maxival)
              maxival = ival[i][j];
         }
      }
```

34

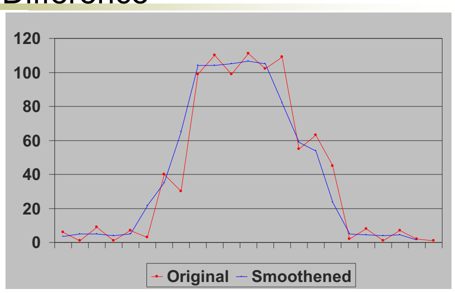
Edge Detection



Graph shows one line of pixel values from an image.
Where did this graph come from?

35

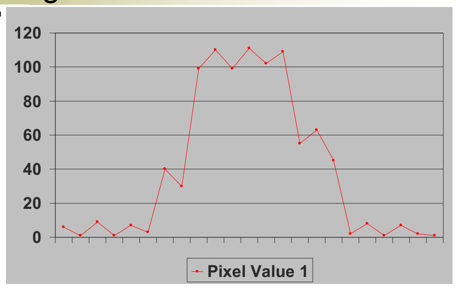
Smoothening before Difference



Smoothening in this case was obtained by averaging two neighboring pixels.

36

Edge Detection



Graph shows one line of pixel values from an image.
Where did this graph come from?

37

Smoothening rationale

- Smoothening: We need to smoothen before we apply the derivative convolution.
- We mean read an image, smoothen it, and then take it's gradient.
- Then apply the threshold.

38

The Four Ones

- The way we will take an average of four neighboring pixels is to convolve the pixels with $\begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}$

Convolving with this is equal to a + b + c + d divided by 4.
(Where a,b,c,d are the four neighboring pixels.)

39

Four Ones, cont.

$$\begin{bmatrix} ¼ & ¼ \\ ¼ & ¼ \end{bmatrix} \text{ Can also be written as: } 1/4 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Meaning now, to get the complete answer, we should compute:

$$1/4\left(\left[\text{ Image }\right] \circledast \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right) \circledast \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix}$$

40

Four Ones, cont.

$$1/4\left(\left[\text{ Image }\right] \circledast \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right) \circledast \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix}$$

By the associative property of convolution we get this next step.

$$1/4 \left[\text{ Image }\right] \circledast \left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \circledast \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix}\right)$$

We do this to call the convolution code only once, which precomputes
The quantities in the parentheses.

41

Four Ones, cont.

$$1/4 \left[\text{ Image }\right] \circledast \left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \circledast \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix}\right)$$

We will be **getting rid** of the 1/4 factor, because it turns out that when we forget about it, to fix our forgetfulness, we merely need to raise our threshold by a factor of 4 (which is O.K. because we were quite arbitrary about how to pick a threshold).

42

Four Ones, cont.

$$\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \circledast \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix} \right)$$

As we did in the convolution slide:

We combine this to get:

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

This table is a result of doing a scanning convolution.

43

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

Gx

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

Gy

First these two masks are applied to the image.

The magnitude of the gradient is then calculated using the formula which we have seen before:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Then, the two output tables of the masks and image are combined using the magnitude formula. This gives us a smoothened gradient magnitude output.

44

Sobel Algorithm…another way to look at it.

- The Sobel algorithm uses a smoothener to lessen the effect of noise present in most images. This combined with the Roberts produces these two – 3X3 convolution masks.

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

G_x

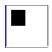
+1	+2	+1
0	0	0
-1	-2	-1

G_y

45

Step 1 – use small image with only black (pixel value = 0) and white (pixel value = 255)

20 X 20 pixel image of black box on square white background



This is the original image. Nothing has been applied to it yet

Pixel values for above image

255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

46

Step 2 – Apply Sobel masks to the image, first the x and then the y.

X mask values

-255	-255	0	0	0	0	0	255	255	0	0	0	0	0	0	0	0	0
-765	-765	0	0	0	0	0	765	765	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-765	-765	0	0	0	0	0	765	765	0	0	0	0	0	0	0	0	0
-255	-255	0	0	0	0	0	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The masks are then convolved with the x and y masks. These are the results of the convolution

Y mask values

255	765	1020	1020	1020	1020	1020	765	255	0	0	0	0	0	0	0	0	0
255	765	1020	1020	1020	1020	1020	765	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-255	-765	-1020	-1020	-1020	-1020	-1020	-765	-255	0	0	0	0	0	0	0	0	0
-255	-765	-1020	-1020	-1020	-1020	-1020	-765	-255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

47

Step 3 – Find the Magnitudes using the formula c = sqrt(X^2 + Y^2)

X mask

-255	-255	0	0	0	0	0	255	255	0	0	0	0	0	0	0	0	0
-765	-765	0	0	0	0	0	765	765	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-1020	-1020	0	0	0	0	0	1020	1020	0	0	0	0	0	0	0	0	0
-765	-765	0	0	0	0	0	765	765	0	0	0	0	0	0	0	0	0
-255	-255	0	0	0	0	0	255	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

^ Using the formula above, the X mask and the Y mask of the image is combined to create the magnitude image below. This is applied to each individual pixel.

Magnitudes

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	85	190	240	240	240	240	240	190	85	0	0	0	0	0	0	0	0	0	0
0	190	255	240	240	240	240	240	255	190	0	0	0	0	0	0	0	0	0	0
0	240	240	0	0	0	0	0	240	240	0	0	0	0	0	0	0	0	0	0
0	240	240	0	0	0	0	0	240	240	0	0	0	0	0	0	0	0	0	0
0	240	240	0	0	0	0	0	240	240	0	0	0	0	0	0	0	0	0	0
0	240	240	0	0	0	0	0	240	240	0	0	0	0	0	0	0	0	0	0
0	240	240	0	0	0	0	0	240	240	0	0	0	0	0	0	0	0	0	0
0	240	240	0	0	0	0	0	240	240	0	0	0	0	0	0	0	0	0	0
0	190	255	240	240	240	240	240	255	190	0	0	0	0	0	0	0	0	0	0
0	85	190	240	240	240	240	240	190	85	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

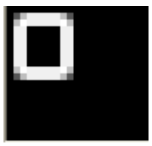
Y mask

255	765	1020	1020	1020	1020	1020	765	255	0	0	0	0	0	0	0	0	0
255	765	1020	1020	1020	1020	1020	765	255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-255	-765	-1020	-1020	-1020	-1020	-1020	-765	-255	0	0	0	0	0	0	0	0	0
-255	-765	-1020	-1020	-1020	-1020	-1020	-765	-255	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

48

Step 4 – Apply threshold, say 150, to the combined image to produce final image.

Before threshold



The left image is the image combined with masks. The right image uses a threshold to separate the grey into either 0 or 255.

After threshold of 150

These are the edges it found

49

Canny Algorithm

Part One

Convove with Gaussian instead of four 1's

Four 1's is hat or box function, so preserves some kinks due to corners

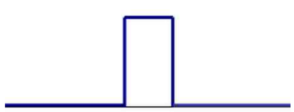
When convolving, if the picture is the table on the left, then the table on the right can be viewed as a Filter. i.e., the resulting output is going to be whatever parts of the picture (table on the left) are to be emphasized. Thus, table on the right is like a guard, permitting only certain parts of the input to pass through to the resulting output.

50

Canny Algorithm

Part One

Four 1's is hat or box function, so preserves some kinks due to corners

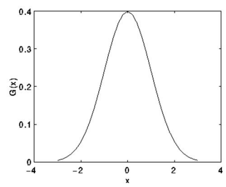


51

Canny Algorithm, Part One

Gaussian $$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

The bell curve of the Guassian function makes it ideal to smoothen an image, with out having kinks in the output.



52

Canny Algorithm, Part One

Using Gaussian

Instead of convolving with +1|-1 for derivative, take derivative of Gaussian

This is the derivative of a one dimensional function. For an image, we will need to use a two dimensional function.

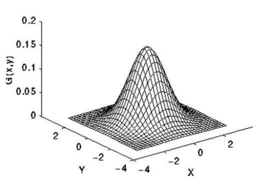
$$\frac{dg_1}{dx} = -\frac{x}{\sigma^2} g_1(x)$$

$$= -xe^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2}$$

53

Canny Algorithm, Part One

2-d Gaussian $$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Take the derivative of this function and use it in our program. Plug in values for x, y, and sigma, to give us a table of weights.

54

Canny Algorithm, Part One

For derivative, should we convolve Gaussian table with a table of +1/-1's (as Sobel did) or should we use a formula that has already taken the derivative of the Gaussian? In both cases, we would have to generate a table from a formula. In the first option, the table would be generated from the Gaussian formula, and in the second option, the table would be generated from the (Gaussian's) derivative equation.

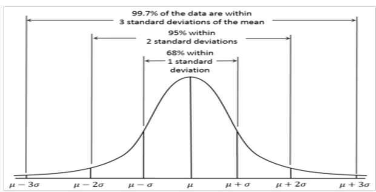
It turns out that the second option is better, if the intent is to increase accuracy. In either case, we would need to figure out how large our table should be. We will discuss this size calculation in the context of the first option, but the reasoning will carry over to the second option as well.

55

Canny Algorithm, Part One

Using Gaussian, how wide will it be?

We start this size calculation, by noticing this:

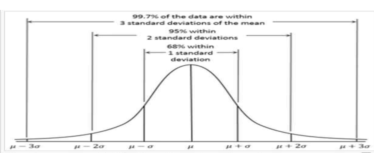


For the normal distribution, the values less than one standard deviation away from the mean account for 68.27% of the set; while two standard deviations from the mean account for 95.45%; and three standard deviations account for 99.73%.

56

Canny Algorithm, Part One

This figure is from Wikipedia



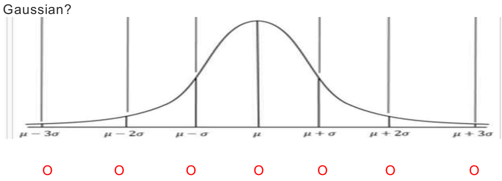
For the normal distribution, the values less than one standard deviation away from the mean account for 68.27% of the set; while two standard deviations from the mean account for 95.45%; and three standard deviations account for 99.73%.

We notice that the Gaussian function has very small values to the left of the third negative Standard Deviation, and the same behavior to the right of the third positive Standard Deviation. Hence, we only need to represent the Gaussian within this window, from negative third SD to positive third SD. Outside of this window, the values of the function are tiny, and hence negligible, and so we choose to crop the function at these ends.

57

Canny Algorithm, Part One

How wide to represent the Gaussian?



So, we will sample our Gaussian at the location of these circles. In general, Our Gaussian will be represented at one position for the center, and (plus) however many positions are needed to cover 3σ on each side of the center. A boss will tell us how large σ needs to be, i.e., how wide the Gaussian is to be.

58

Canny Algorithm, Part One

Gaussian's width:

The boss tells us how wide each sigma is to be.
So, if boss says σ is to be 1 pixel wide, then the total window will be

1 + 3*1 + 3*1, i.e., 7 pixels wide.

If boss says σ is to be 2, then window's total width is

1 + 3*2 + 3*2, i.e., 13 pixels wide.

If σ is to be 3, then total width is 19. And so on. The basic idea is that the width is give by: 1 + 6*σ.

59

Canny Algorithm, Part One

Using Gaussian

We can use either a table of weights, or we can use an equation within the code to Increase accuracy.

$\frac{1}{273}$

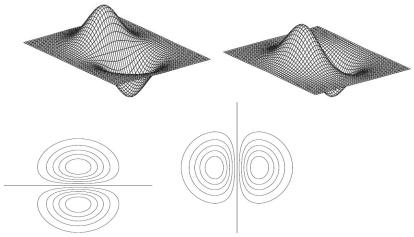
1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

60

Canny Algorithm, Part One

2-d Gaussian, plus derivative

Instead of convolving with +1|-1 for derivative, take derivative of Gaussian



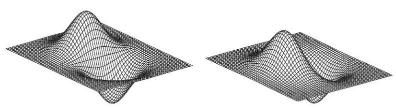
$$|G| = \sqrt{Gx^2 + Gy^2}$$

61

Canny Algorithm, Part One

2-d Gaussian, plus derivative

Instead of convolving with +1|-1 for derivative, take derivative of Gaussian



$$\frac{\partial G_\sigma}{\partial x} = x e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$|G| = \sqrt{Gx^2 + Gy^2}$$

62

Canny Algorithm, Part One

So, the main difference between Canny Part One and Sobel is the smoothener (Canny uses a Gaussian Sobel uses the four one's.

To write code for canny, we will start with marrh.c and do these steps.

- -- Marrh.c uses flexible size masks (which we need), we will keep this part of the code.
- -- Marrh.c uses second derivatives, whereas we need only first derivatives (we need first x- and y- derivatives), so we will change the equation in the marrh.c line to be the first x-derivative.
- -- Then, because we need two derivatives, we will double up on that line, i.e., make a copy of it to compute the y-derivative, finally ending up with two masks (xmask and ymask).

63

Canny Algorithm, Part One

-- Then use the convolution code from marrh but remember to double up on it, to get two outputs.

-- Then delete the code in marrh that is below the convolution code.

-- Then bring in the sqrt (of squares) code from sobel. This will compute the magnitude, will scale it for output, and will print it out.

-- At this point, you are done with Canny part One, and your code should produce output very similar to the Sobel magnitude image.

64

Canny Algorithm, Part Two

Peak Finding, Non-Maxima Suppression

Consider four directions available in 3x3 neighborhood

Peak finding first determines edge direction, and then tests the pixels that are on the sides (perpendicular) to see if ithe center pixel has the highest value among said neighbors. If it has the highest value then it will be considered a potential edge. So, peaks are potential edges. If a pixel is not a peak, it has no chance of being an edge.

65

Canny Algorithm, Part Three

Double Threshholding, Hysteresis Thresholding

First, accept all peak pixels where Magnitude exceeds HIGH, then all who are connected to HIGHs and also exceed a lower LO threshold.

Often times in edge detection, there will be edges that will be very close to the threshold. This causes noise and dotted edges. Using double thresholding, we can eliminate that noise by having two thresholds, a high and a low. An edge must first exceed the higher threshold. Any peak pixel touching a valid edge must have a value less than the low threshold to not be considered an edge. Conversely, any peak pixel must at least exceed the low threshold for a valid edge neighbor to bring the peak into the final edge.

66

Canny Algorithm, Part Four

Automatically, determine HI and LO.

67