Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 8

Segmentation

Edge detection and boundary-based approaches:
gradient, Laplacian, Laplacian of Gaussian, Canny,
boundary chain codes, contour extraction, snakes

Edge Detection

– An approach to segmentation

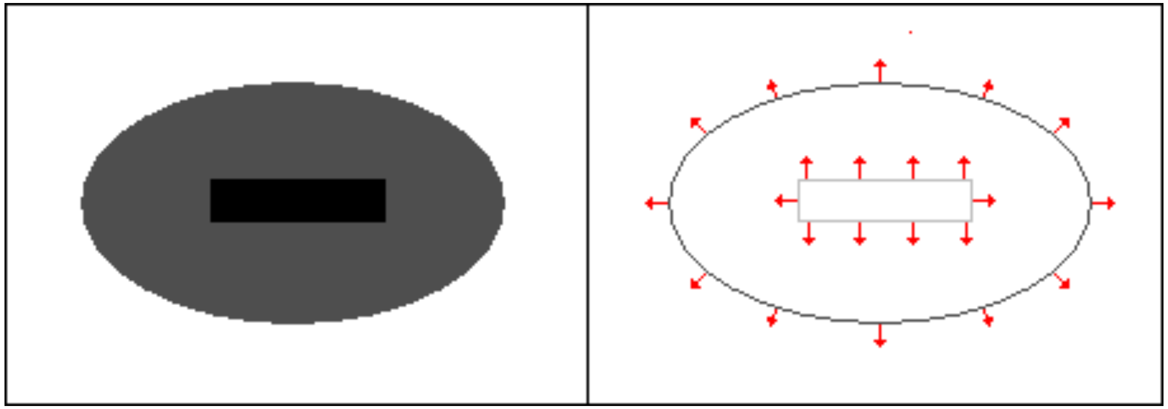
– Based on the analysis of the discontinuities in an image

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Edge Detection

Edges have

- Magnitude
- Direction (Orientation)

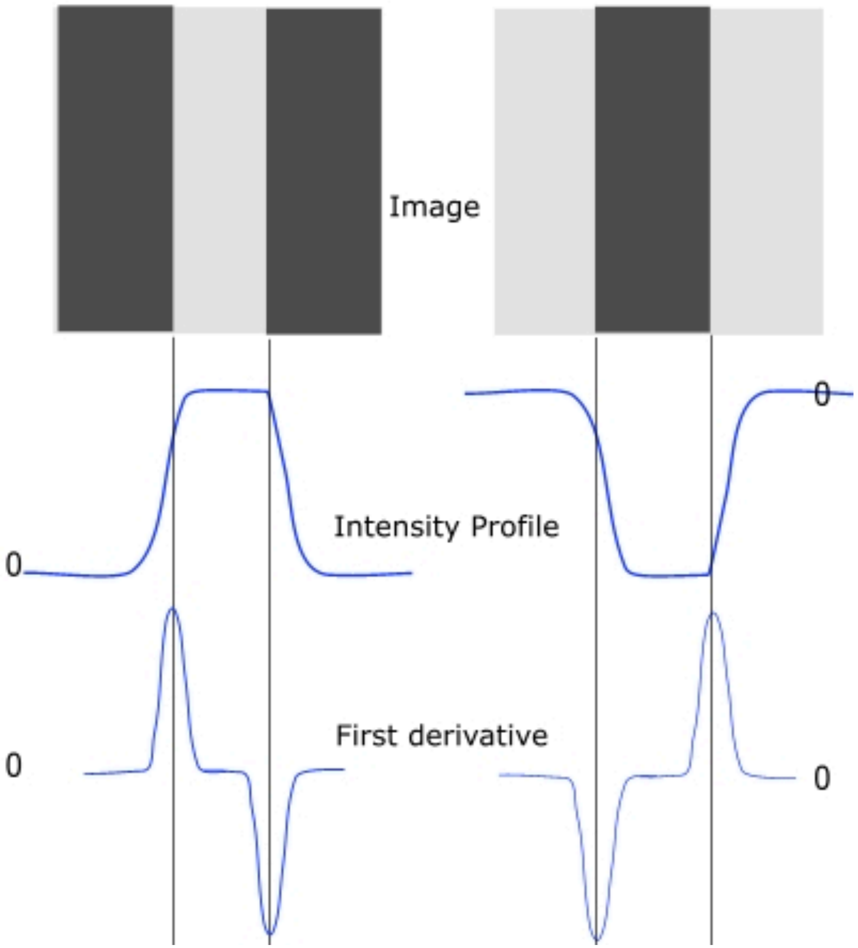


Edge Profiles

- Step
- Real
- Noisy

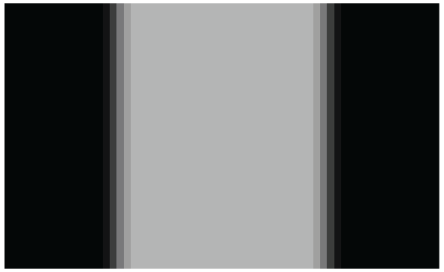
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Edge Detection

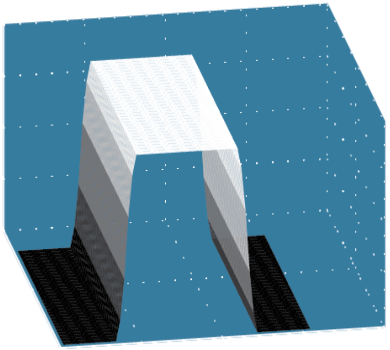


Source: https://mipav.cit.nih.gov/pubwiki/index.php/Edge_Detection:_Zero_X_Laplacian

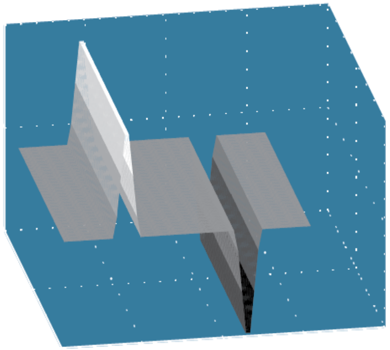
Edge Detection



Image



Intensity profile



First derivative

Edge Detection

- Define a local edge in an image to be a transition between two regions of significantly different intensities

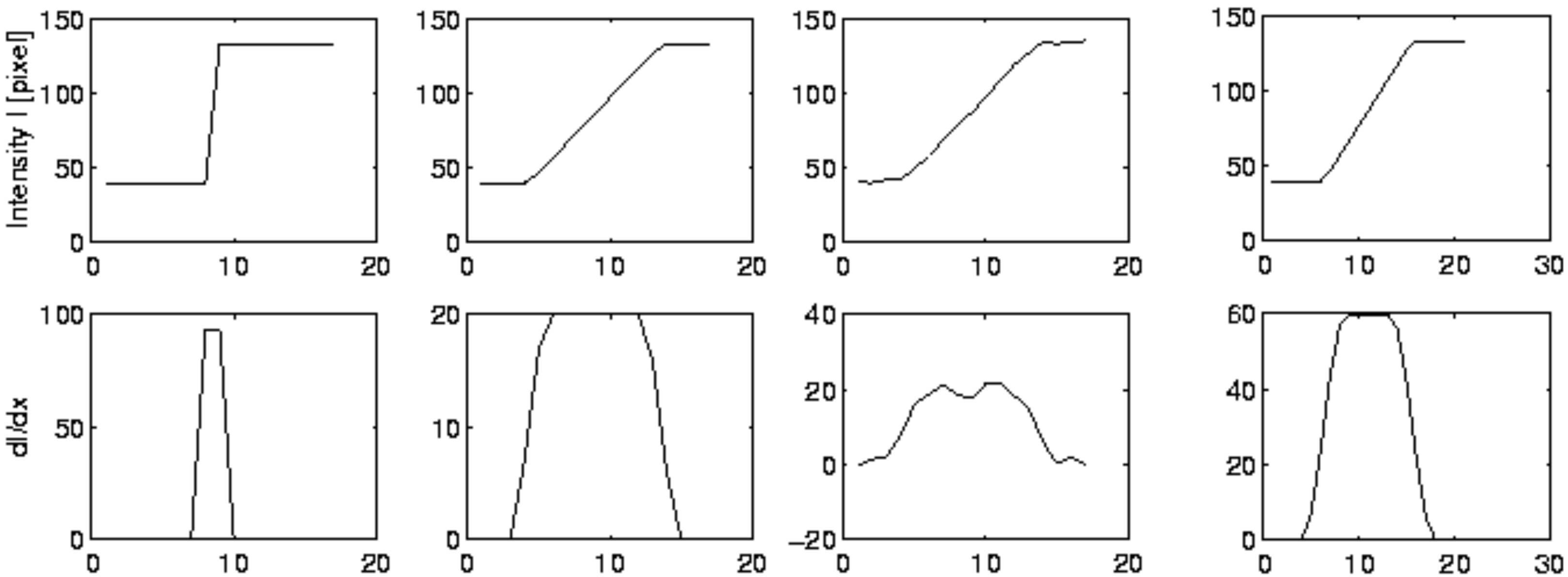
- The gradient function of the image, which measures the rate of change, will have large values in these transitional boundary areas

  - Enhance the image by **estimating** its gradient function
  - An edge is present if the gradient value is greater than some defined threshold

Edge Detection

Signal 1D
= Input

First derivative
= gradient



Perfect edge

Perfect spread edge

Edge with noise
s =1

Edge with noise after filtering (Gaussian)

Credit: Markus Vincze, Technische Universität Wien

Edge Detection

- Gradient functions are easy to understand in the discrete domain of digital images

  Derivatives become simple first differences ($h = 1$)

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- Thus, the first difference of a 1D function is simply

$$f(x+1) - f(x)$$

Edge Detection

Consider the following

1-D discrete (sampled & quantized) signal $f(x)$

its first derivative (i.e. 1st difference) $f'(x)$

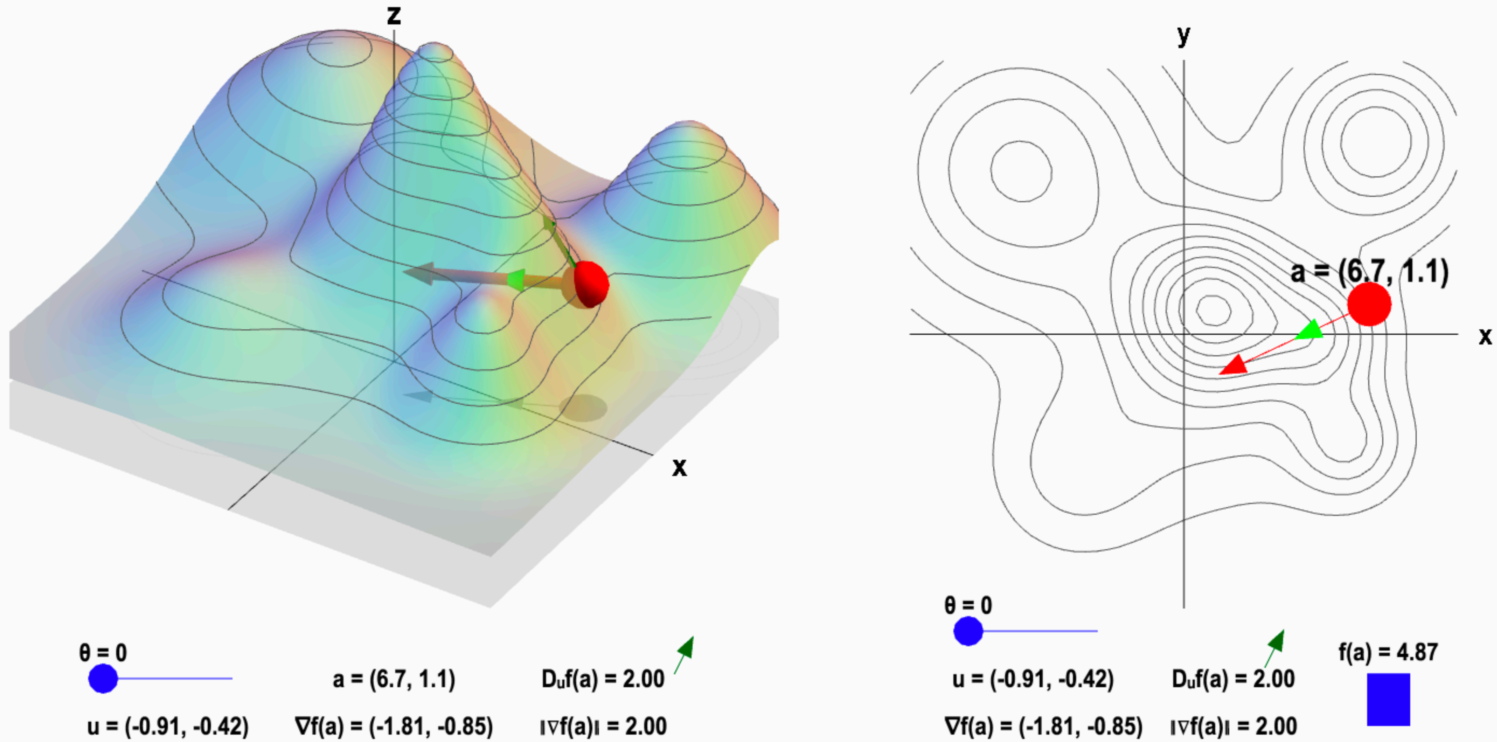
$f(x)$	1	2	2	1	0	1	1	0	1	9	8	9	9	9	8
$f'(x)$	1	0	-1	-1	1	0	-1	1	8	-1	1	0	0	-1	

Edge Detection

In a 2D image, the gradient is a vector

It has a magnitude and a direction

Edge Detection



Gradient and directional derivative on a mountain. The height of a mountain range described by a function $f(x, y)$ is shown as surface plot in three-dimensions (left) and a two-dimensional level curve plot (right). In each panel, a red point can be moved by the mouse to change the location **a** where the gradient $\nabla f(\mathbf{a})$ is calculated. Since f is a function of two variables, the point **a** and the gradient are two-dimensional. The two-dimensional point **a** is illustrated by the shadow of the red point on the xy-plane below the surface plot and by the red point itself on the level curve plot. The two dimensional gradient vector $\nabla f(\mathbf{a})$ is illustrated by the red vector emanating from the red point as well as by its shadow below the surface plot.

Credit: http://mathinsight.org/directional_derivative_gradient_introduction

Edge Detection

In a 2D image, the gradient is a vector

It has a magnitude and a direction

If $\partial/\partial x$ and $\partial/\partial y$ represent the rates of change of a 2D function $f(x, y)$ in the x and y directions respectively, then the rate of change in a direction θ is given by

$$\frac{\partial f}{\partial x}\cos\theta + \frac{\partial f}{\partial y}\sin\theta$$

Edge Detection

- The direction θ at which the rate of change has the greatest magnitude is given by

$$\arctan\left[\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right]$$

- The magnitude is given by $\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

- The gradient of $f(x, y)$ is a **vector** at (x, y) with this magnitude and direction

Edge Detection

- Gradient functions are easy to understand in the discrete domain of digital images

Derivatives become simple first differences ($h = 1$)

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- For example, the first difference of a 2D function in the x direction is simply

$$f(x+1, y) - f(x, y)$$

- Similarly, the first difference of a 2D function in the y direction is simply

$$f(x, y+1) - f(x, y)$$

Edge Detection

The essential differences between all gradient edge detectors are

- the directions which the operators use to estimate the partial derivatives
- the manner in which they approximate the one-dimensional derivatives of the image function in these directions
- the manner in which they combine these approximations to form the gradient magnitude

(a) Roberts

1	0
0	-1

0	1
-1	0

(a)

(b) Sobel

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

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

(b)

(c) Prewitt

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

(c)

Edge Detection

Roberts Cross

Strength: $G = \sqrt{G_x^2 + G_y^2}, \quad G = |G_x| + |G_y|$

Angle: $\theta = \arctan(G_y / G_x) - 3\pi / 4$

+1	0
0	-1

$G_x = \delta I / \delta x$

0	+1
-1	0

$G_y = \delta I / \delta y$

Original image

Edge strength

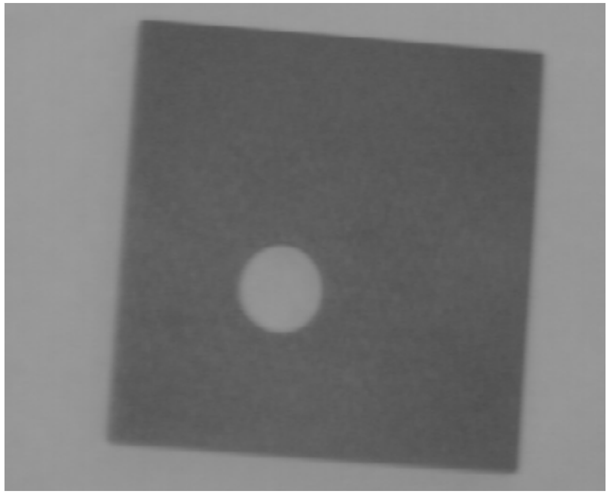
Binary image, threshold:80

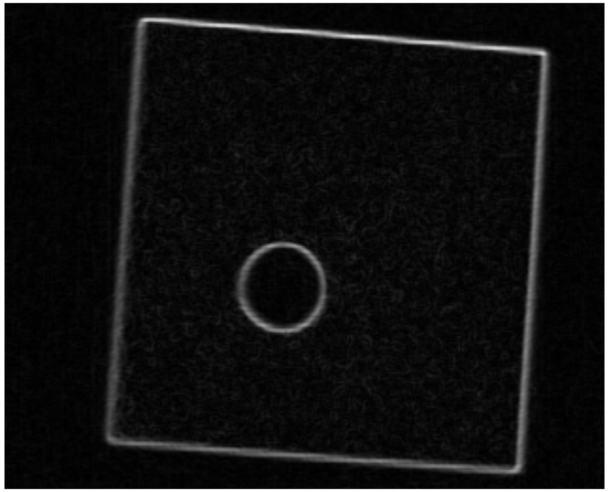
Credit: Markus Vincze, Technische Universität Wien

Edge Detection

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

Angle: $\theta = \arctan(G_y / G_x)$





Sobel

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

Gx

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

Gy

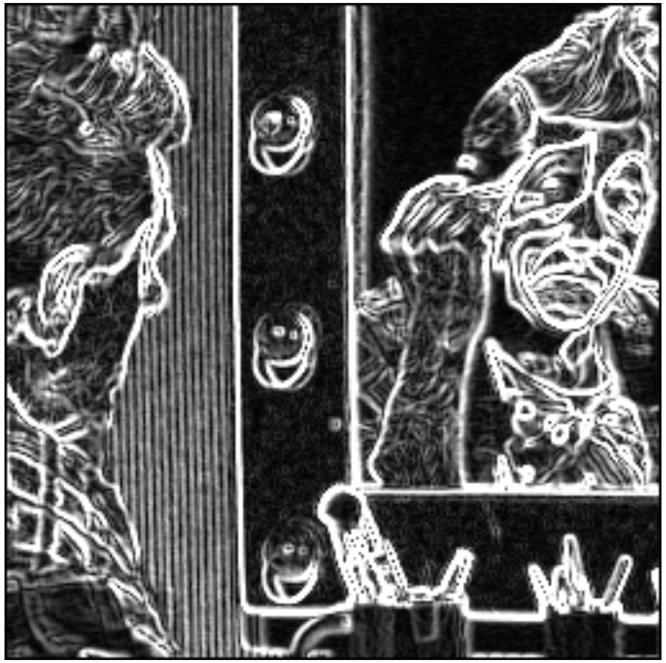
Prewitt

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

Gx

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

Gy



Credit: Markus Vincze, Technische Universität Wien

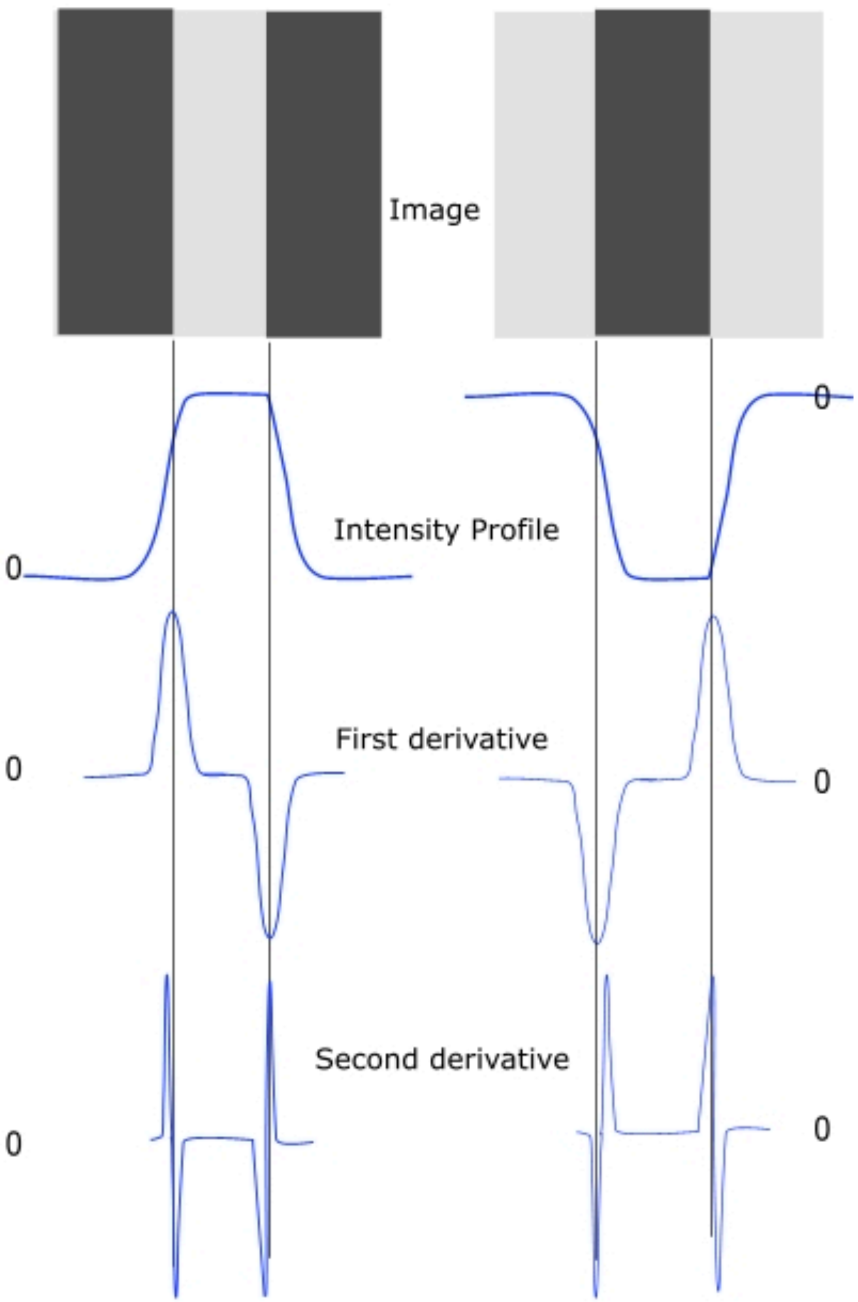
Edge Detection

Edge detection has been discussed on the basis of first derivative directional operators

However, an alternative method uses an approximation to the Laplacian

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

i.e. the sum of second-order, unmixed, partial derivatives



Credit: https://mipav.cit.nih.gov/pubwiki/index.php/Edge_Detection:_Zero_X_Laplacian

Edge Detection

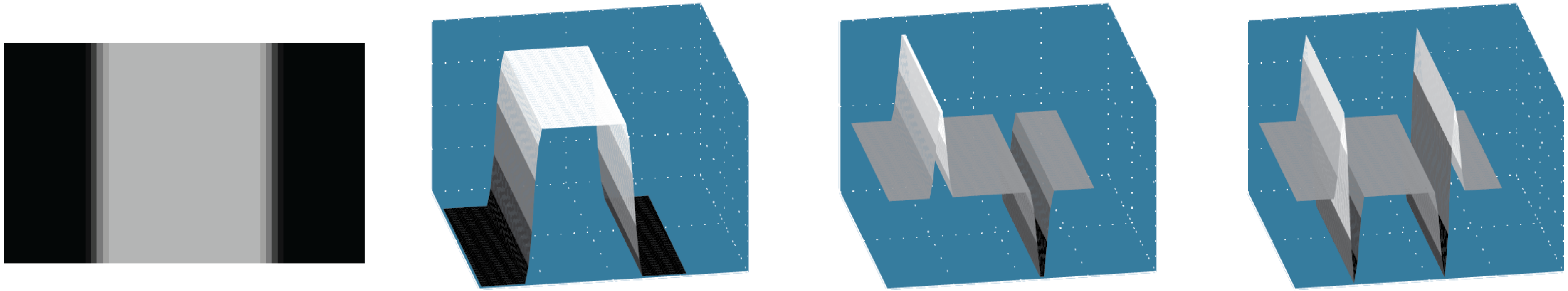


Image | Intensity profile | First derivative | Second derivative

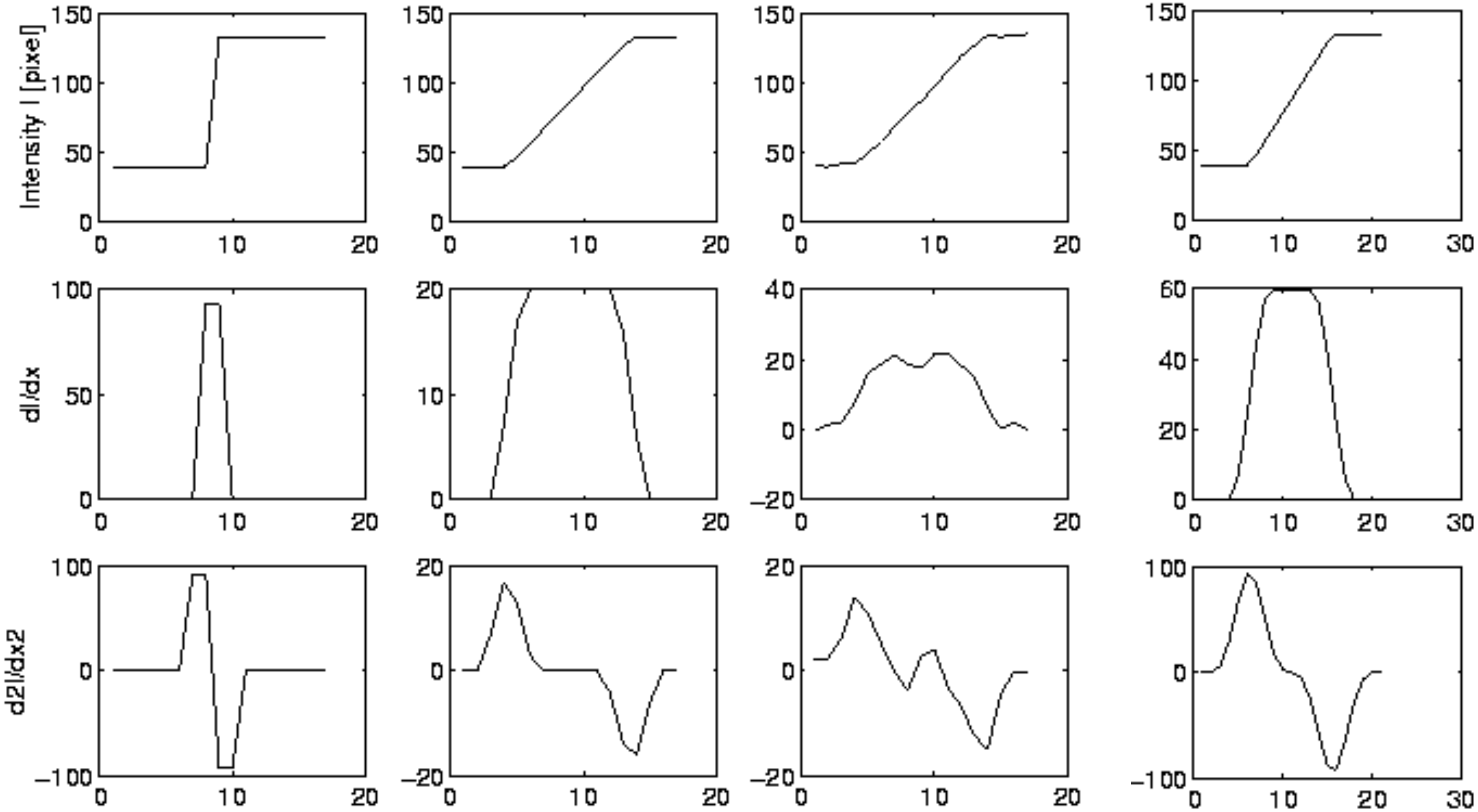
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Edge Detection

Signal 1D
= Input

First derivative
= gradient

Second
derivative
(Laplace
operator)



Perfect edge

Perfect spread
edge

Edge with noise
s =1

Edge with noise after
filtering (Gaussian)

Credit: Markus Vincze, Technische Universität Wien

Edge Detection

- Gradient functions are easy to understand in the discrete domain of digital images

Derivatives become simple first differences ($h = 1$)

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- Thus, the first difference of a 1D function is simply

$$f(x+1) - f(x)$$

- The second difference of a 1D function is simply the first difference of the first difference

$$f(x+1) - 2f(x) + f(x-1)$$

Edge Detection

1st difference at x $\qquad f(x) - f(x\text{-}1)$

1st difference at $x+1$ $\qquad f(x+1) - f(x)$

2nd difference in x =
1st difference of 1st difference: $\qquad (f(x+1) - f(x)) - (f(x) - f(x\text{-}1))$

$$= \quad f(x+1) - 2f(x) + f(x\text{-}1)$$

This is often represented by a convolution or filter kernel

+1 -2 +1

Edge Detection

Consider the following

1-D discrete (sampled & quantized) signal $f(x)$

its first derivative (i.e. 1[st] difference) $f'(x)$

its second derivative (i.e. 2[nd] difference) $f''(x)$

$f(x)$	1	2	2	1	0	1	1	0	1	9	8	9	9	9	8
$f'(x)$	1	0	-1	-1	1	0	-1	1	8	-1	1	0	0	-1	
$f''(x)$		-1	-1	0	2	-1	-1	2	7	-9	2	-1	0	-1	

Edge Detection

The standard 2d approximation is given by:

$$L(x, y) = -4\,f(x, y) + f(x\text{-}1, y) + f(x+1, y) + f(x, y\text{-}1) + f(x, y+1)$$

0	1	0
1	−4	1
0	1	0

1st difference at x	$f(x, y) - f(x\text{-}1, y)$
1st difference at x+1	$f(x+1, y) - f(x, y)$
2nd difference in x =	
1st difference of 1st difference, *i.e.*	$(f(x+1, y) - f(x, y)) - (f(x, y) - f(x\text{-}1, y))$
2nd difference in x =	$f(x+1, y) - 2f(x, y) + f(x\text{-}1, y)$
2nd difference in y =	$f(x, y+1) - 2f(x, y) + f(x, y\text{-}1)$
Laplacian = sum of 2nd differences =	$-4\,f(x, y) + f(x\text{-}1, y) + f(x+1, y) + f(x, y\text{-}1) + f(x, y+1)$

Edge Detection

The digital Laplacian has

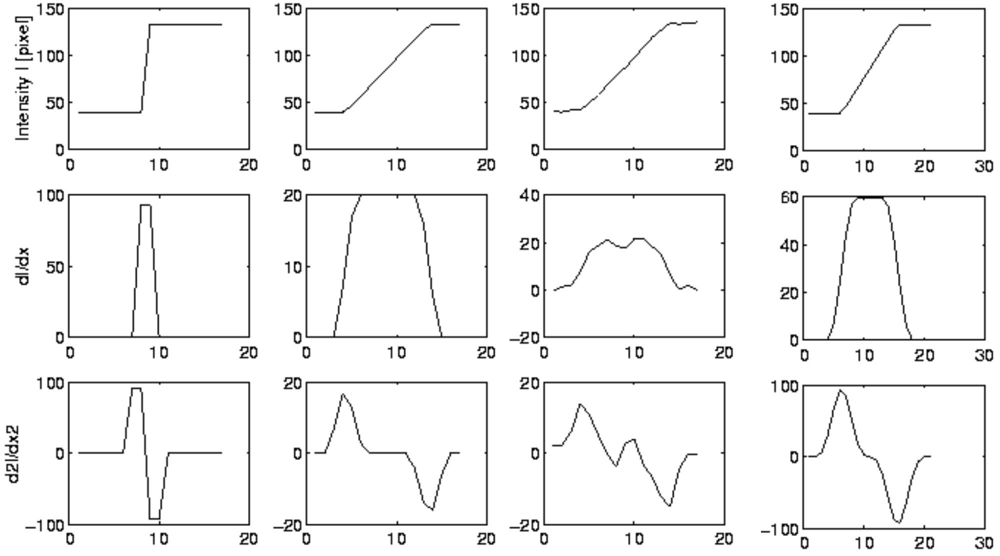
- zero-response to linear ramps (and thus gradual changes in intensity)
- but it does respond on either side of the edge, once with a positive sign and once with a negative sign

Edge Detection

To detect edges

– the image is enhanced by evaluating the digital Laplacian

– isolating the points at which the resultant image goes from positive to negative, *i.e.*, at which it crosses zero



Credit: Markus Vincze, Technische Universität Wien

Edge Detection

Marr-Hildreth edge detector (Laplacian of Gaussian)

– First smooth the image by convolving it with a two-dimensional Gaussian function

– and then isolate the zero-crossings of the Laplacian of this image

$$\nabla^2 \left\{ I(x, y) * G(x, y) \right\}$$

Edge Detection

Why do we convolve the image with a Gaussian function?

- The Gaussian blurs the image
- This wipes out all structure at scales much smaller than the space constant σ (standard deviation) of the Gaussian
- Thus, we can select the spatial scale at which we process the image

D. Marr, Vision, Freeman Press, 1982

Edge Detection

Why the Gaussian?

- The Gaussian has the desirable characteristic of being smooth and localized in both the spatial and frequency domains

- It is the unique distribution that is simultaneously optimally localized in both domains

- Thus it is least likely to introduce any changes that were now present in the original image

D. Marr, Vision, Freeman Press, 1982

Edge Detection

The evaluation of the Laplacian and the convolution commute so that, for a Gaussian with a given standard deviation, we can derive a single filter

The Laplacian of Gaussian

$$\nabla^2 \left\{ I(x,y) * G(x,y) \right\} = \boxed{\nabla^2 G(x,y)} * I(x,y)$$

Edge Detection

Furthermore, this 2D convolution is separable into four 1D convolutions

$$\nabla^2 \left\{ I(x,y) * G(x,y) \right\} =$$

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

Edge Detection

Laplacian

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

0	1	0
1	−4	1
0	1	0

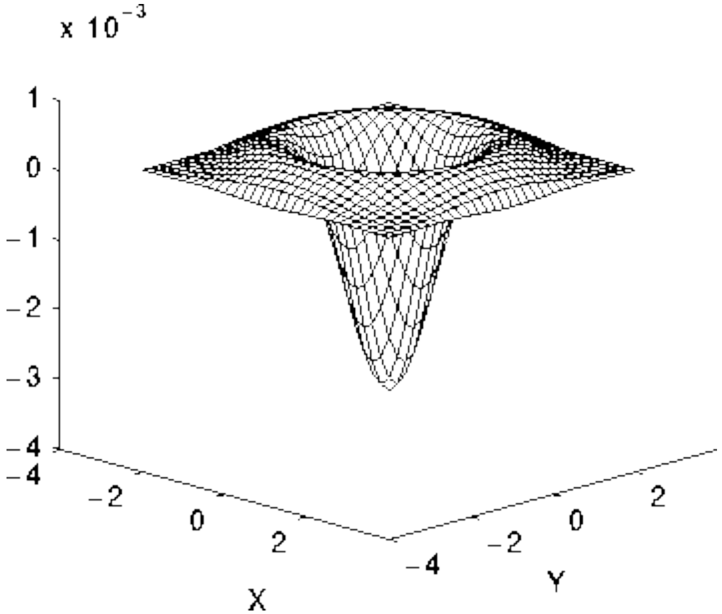
1	1	1
1	−8	1
1	1	1

Credit: Markus Vincze, Technische Universität Wien

Edge Detection

Laplacian of Gaussian

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left(1-\frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$



LoG or Mexican Hat

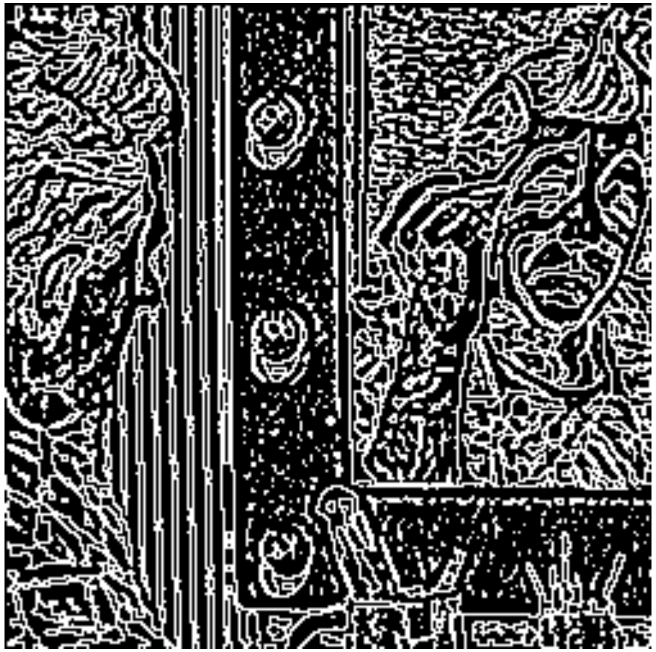
0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Example filter 9x9, σ =1.4

Credit: Markus Vincze, Technische Universität Wien

Edge Detection

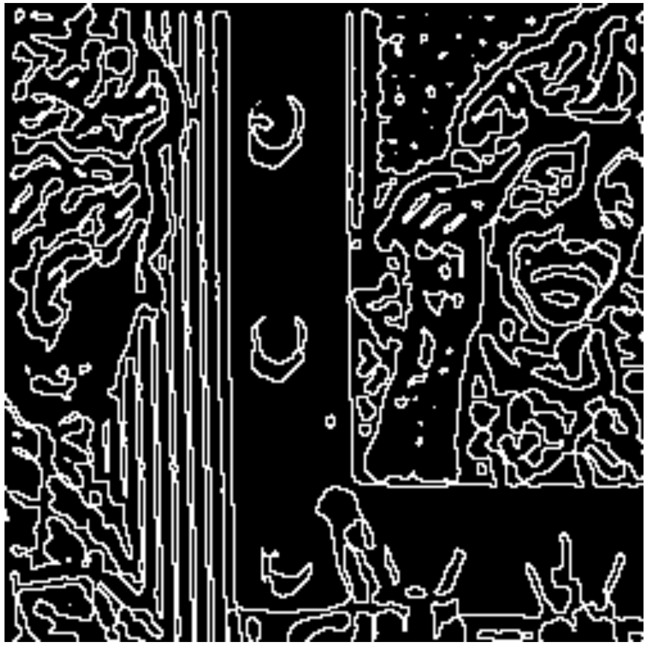
LoG with $\sigma = 1$



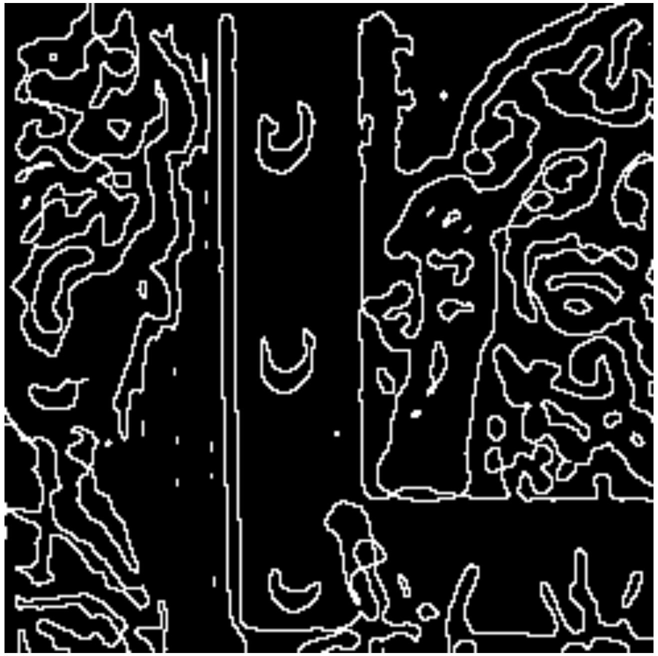
All zero crossings

Credit: Markus Vincze, Technische Universität Wien

Edge Detection



Zero crossings at $\sigma = 2$



Zero crossings at $\sigma = 3$

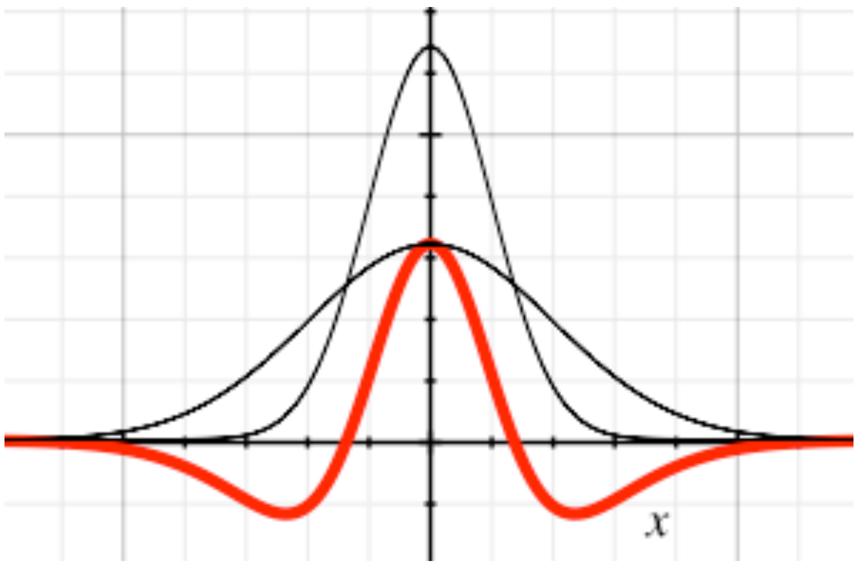
$\sigma = 1$, strong zero crossings (difference to neighbours >40)

Credit: Markus Vincze, Technische Universität Wien

Edge Detection

The Laplacian of Gaussian function is well approximated by the Difference of Gaussian function

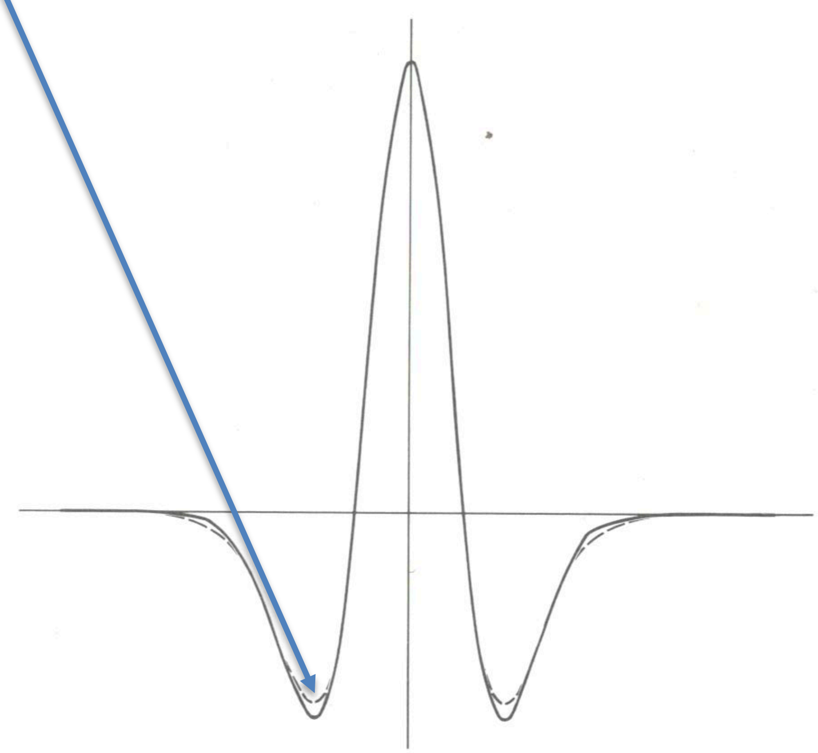
Difference of two different Gaussian functions: $\sigma_1 = 1.6\ \sigma_2$



Credit: http://bigwww.epfl.ch/teaching/iplablibrary/filtering4/index.php

Edge Detection

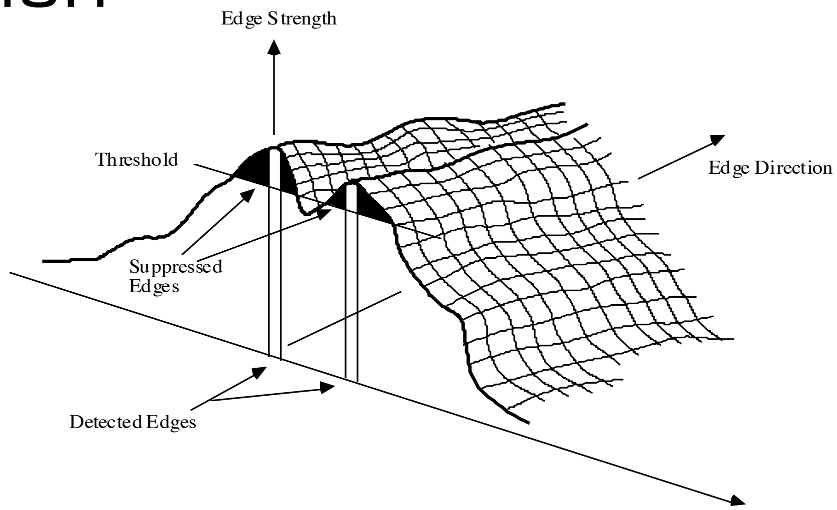
The Laplacian of Gaussian function is well approximated by the Difference of Gaussian function



Credit: D. Marr, Vision, Freeman Press, 1982

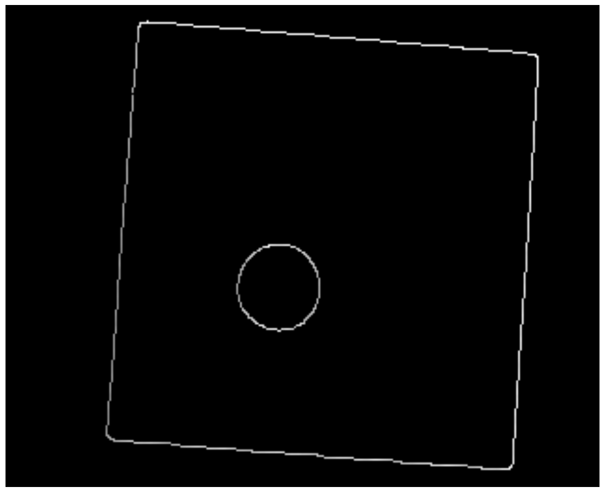
Edge Detection

Canny Edge Detector



1. Gaussian smoothing
2. Gradient estimation
3. Ridge following with non-maxima suppression and hysteresis ($t_2>t_1$)

- Optimised, standard method
- Good compromise
- Thin, one (1) pixel edge (ridge)
- Smoothing eliminates detail

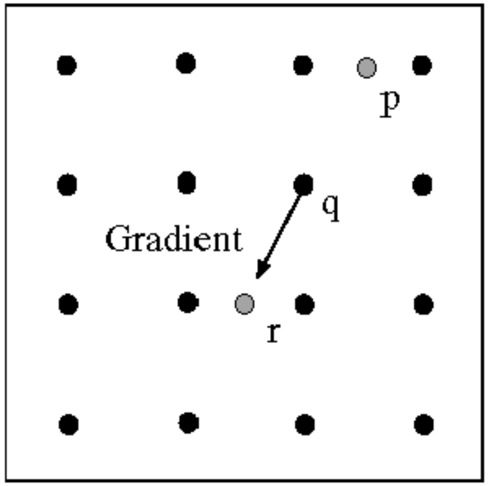


$\sigma = 1, t_2=255, t_1=1$

Edge Detection

Canny Edge Detector

- Identify ridge pixels: maxima in gradient direction
- Value q is a maximum if larger than both p and r
- Interpolate to find p and r

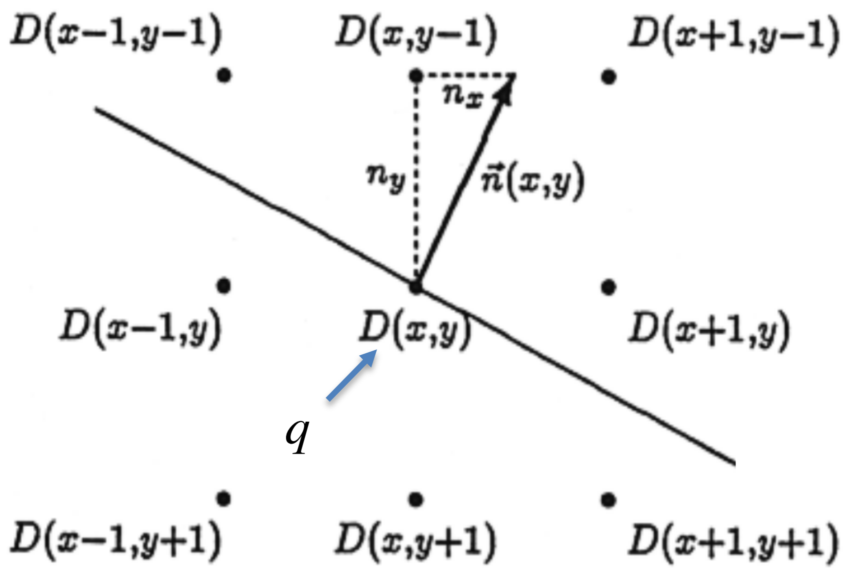


Credit: Markus Vincze, Technische Universität Wien

Edge Detection

Equations for finding the maxima

Notation of pixels:



Interpolation equations:

p → $$D_1 = \frac{n_x}{n_y} D(x+1, y-1) + \frac{n_y - n_x}{n_y} D(x, y-1)$$

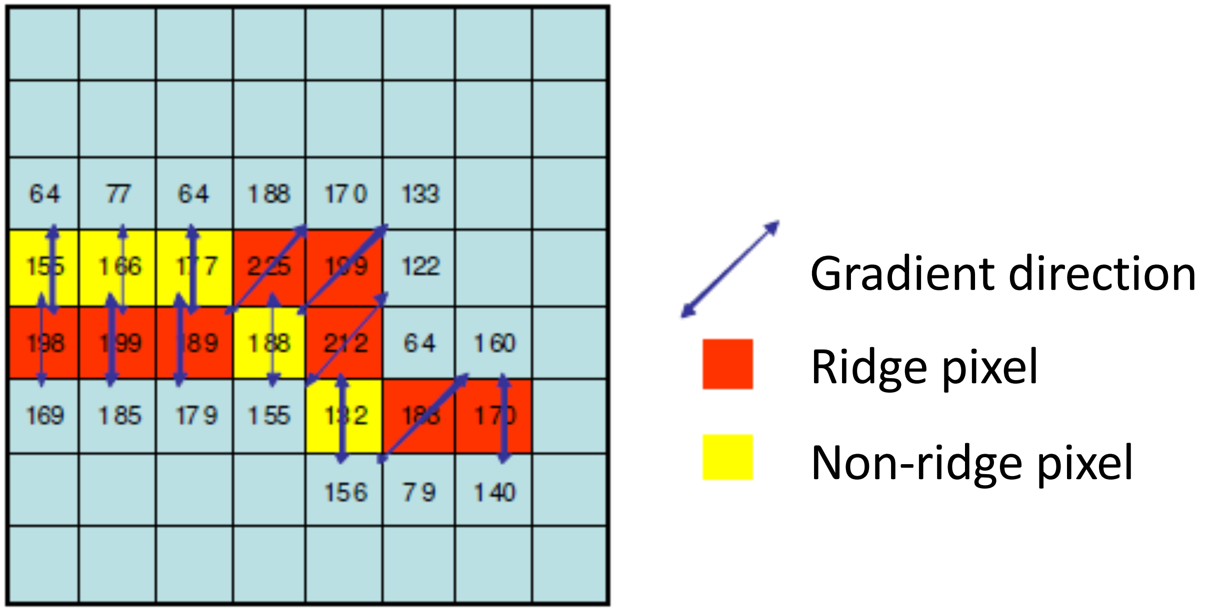
r → $$D_2 = \frac{n_x}{n_y} D(x-1, y+1) + \frac{n_y - n_x}{n_y} D(x, y+1)$$

Credit: Markus Vincze, Vienna University of Technology

Edge Detection

Canny edge detector

– non-maxima suppression



Credit: Markus Vincze, Vienna University of Technology

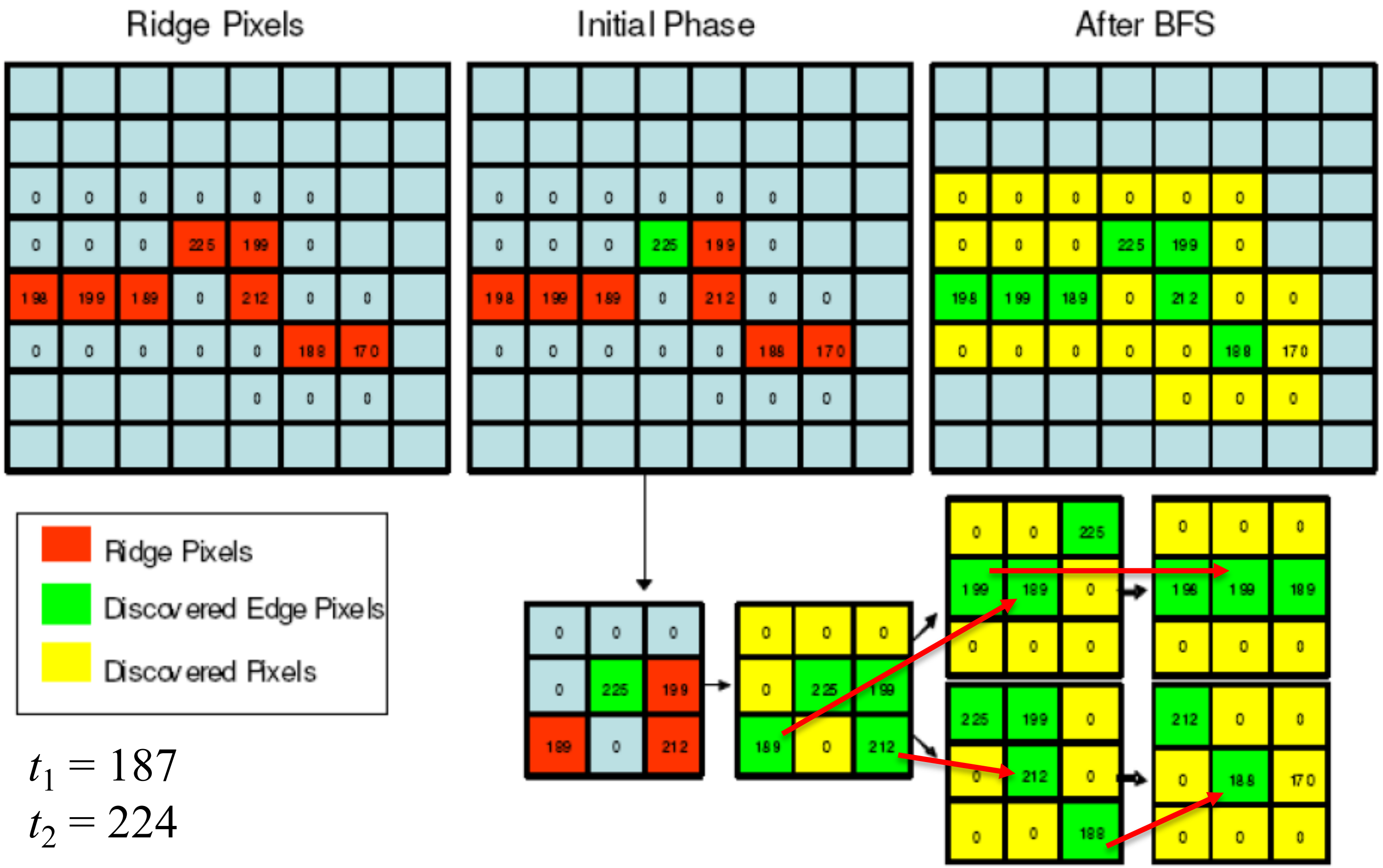
Edge Detection

Hysteresis Thresholding

- 2 thresholds for filtering edges:
  - Low threshold t_1
  - High threshold $t_2 > t_1$
- For all ridge pixels with magnitude $m \geq t_2$: mark as edge pixel
- $t_1 < m < t_2$ and adjacent to an edge pixel: mark as edge pixel
- Identifies strong edges and eliminate weak edges with $m < t_1$

Credit: Markus Vincze, Vienna University of Technology

Edge Detection



$$t_1 = 187$$
$$t_2 = 224$$

Credit: Markus Vincze, Vienna University of Technology

Edge Detection

Hysteresis Algorithm

- Identify and mark all ridge pixels having $m \geq t_2$ as visited edges
- Add pixel into queue Q
- Run a breadth first search (BFS) on Q
  - For each pixel i in Q
    - For each unvisited adjacent pixel j of i
      - Mark j as visited
      - If $m(j) > t_1$, mark j as an edge and add j to Q
    - Remove i from Q
  - Terminate when Q is empty

Credit: Markus Vincze, Vienna University of Technology

Edge Detection

Canny Edge Detector

- Y-Effect: 3 edges meeting in a point are not connected
- Adaptive: detail and edge elements, but image dependent

$\sigma = 1, t_2 = 255, t_1 = 1$

$\sigma = 1, t_2 = 255, t_2 = 220$

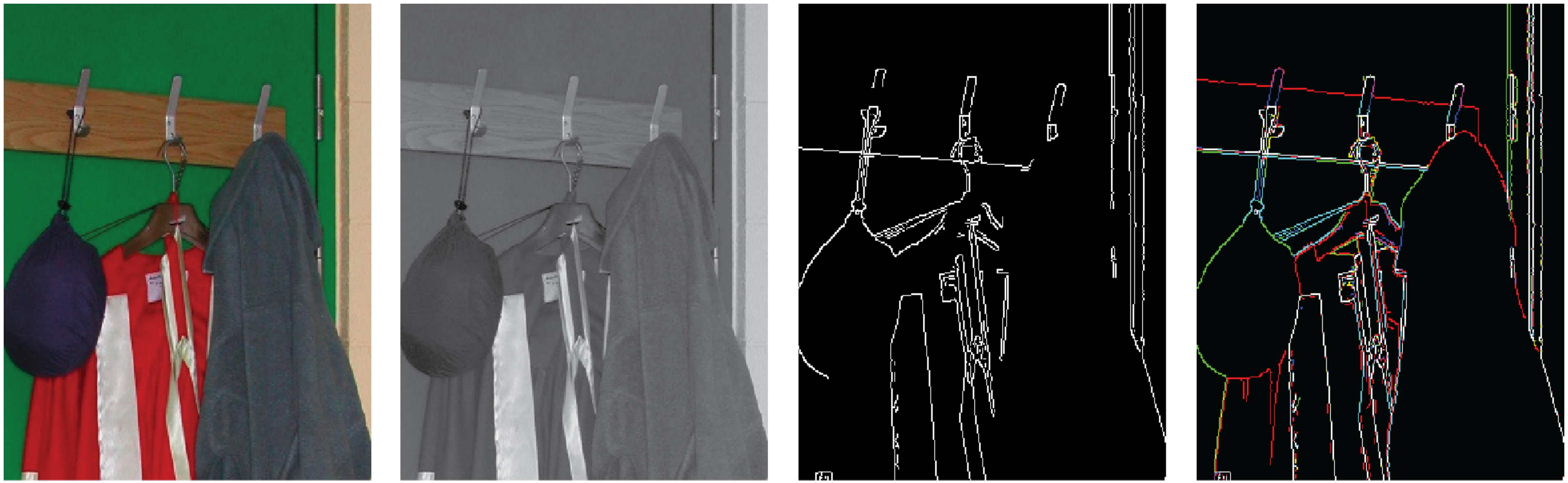
$\sigma = 2, t_2 = 255, t_1 = 1$

Credit: Markus Vincze, Vienna University of Technology

Edge Detection

Multi-spectral edge detection

- Detect edges separately in each spectral band
- Use maximal value OR some linear combination
- Use a different colour model
  - e.g. HLS space with linear combination of H & L



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Boundary Detection

- Edge detection is just the first stage of the boundary-based segmentation process

- Also to aggregate these local edge elements
  - into structures better suited to the process of interpretation

- Normally achieved using processes such as
  - edge thinning (gradient-based edge operators produce thick edges)
  - edge linking
  - gap filling
  - curve-segment linking

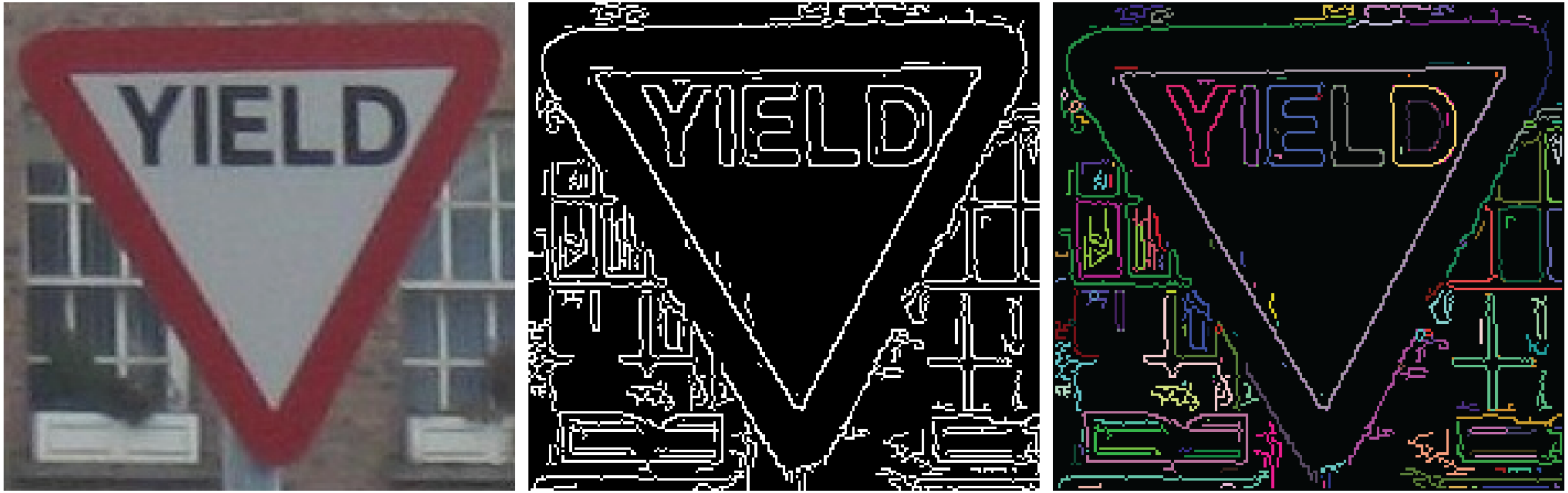
Boundary Detection

- Techniques vary in the amount of knowledge or domain-dependent information that is used in the grouping process

- In order of decreasing use of domain-dependent information
  - Boundary Refining
  - The Hough Transform
  - Graph Search
  - Dynamic Programming
  - Contour Following

Boundary Detection

Representation of Boundaries



In OpenCV, each individual contour is stored as a vector of points and all the contours are stored as a vector of contours (i.e. a vector of vector of points)

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Demos

The following code is taken from the sobelEdgeDetection project in the lectures directory of the ACV repository

See:

```
sobelEdgeDetection.h
sobelEdgeDetectionImplementation.cpp
sobelEdgeDetectionApplication.cpp
```

```
/*
 * function sobelEdgeDetection
 * Trackbar callback - threshold user input
*/

void sobelEdgeDetection(int, void*) {

   extern Mat inputImage;
   extern int thresholdValue;
   extern char* magnitude_window_name;
   extern char* direction_window_name;
   extern char* edge_window_name;

   Mat greyscaleImage;
   Mat edgeImage;
   Mat horizontal_partial_derivative;
   Mat vertical_partial_derivative;
   Mat l2norm_gradient;
   Mat orientation;

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }
   /*****************************************************************************************************/
   /*
    * This code is provided as part of "A Practical Introduction to Computer Vision with OpenCV"
    * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.  All rights reserved.
    */

   Sobel(greyscaleImage,horizontal_partial_derivative,CV_32F,1,0);
   Sobel(greyscaleImage,vertical_partial_derivative,CV_32F,0,1);
   cartToPolar(horizontal_partial_derivative,vertical_partial_derivative,l2norm_gradient,orientation);
   Mat l2norm_gradient_gray = convert_32bit_image_for_display( l2norm_gradient );
   Mat l2norm_gradient_mask, display_orientation;
   l2norm_gradient.convertTo(l2norm_gradient_mask,CV_8U);
   threshold(l2norm_gradient_mask,edgeImage,thresholdValue,255,THRESH_BINARY); // DV thresholdValue edgeImage
   orientation.copyTo(display_orientation, edgeImage);
   Mat orientation_gray = convert_32bit_image_for_display(display_orientation, 0.0, 255.0/(2.0*PI) );

   imshow(magnitude_window_name, l2norm_gradient_gray); // DV
   imshow(direction_window_name, orientation_gray);     // DV
   imshow(edge_window_name, edgeImage);                 // DV
}
```

Demos

The following code is taken from the laplacianOfGaussian project in the lectures directory of the ACV repository

See:

```
laplacianOfGaussian.h
laplacianOfGaussianImplementation.cpp
laplacianOfGaussianApplication.cpp
```

```
/*
 * function laplacianOfGaussian
 * Trackbar callback - Gaussian standard deviation input from user
*/

void laplacianOfGaussian(int, void*) {

   extern Mat src;
   extern int gaussian_std_dev;
   extern char* log_window_name;
   extern char* zc_window_name;
   extern char* mod_zc_window_name;

   int filter_size;

   filter_size = gaussian_std_dev * 4 + 1;  // multiplier must be even to ensure an odd filter size as required by OpenCV
                                            // this places an upper limit on gaussian_std_dev of 7 to ensure the filter size < 31
                                            // which is the maximum size for the Laplacian operator
/***************************************************************************************************/
/*
 * This code is provided as part of "A Practical Introduction to Computer Vision with OpenCV"
 * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.  All rights reserved.
 */

   Mat horizontal_partial_derivative, vertical_partial_derivative;
   Mat laplacian;
   Mat blurred_image1_gray;
   Mat abs_gradient;
   Mat the_gradient;
   GaussianBlur(src,blurred_image1_gray,Size(filter_size,filter_size),(double)gaussian_std_dev); // David Vernon: changed Size() and sigma argume
   Laplacian(blurred_image1_gray,laplacian,CV_32F,filter_size);                                 // David Vernon: changed kernel argument from 3
   Mat zero_crossings;
   Mat modulated_zero_crossings;
   FindZeroCrossings(laplacian,zero_crossings);
   Sobel(blurred_image1_gray,horizontal_partial_derivative,CV_32F,1,0);
   Sobel(blurred_image1_gray,vertical_partial_derivative,CV_32F,0,1);
   abs_gradient = abs(horizontal_partial_derivative) + abs(vertical_partial_derivative);
   abs_gradient.convertTo(the_gradient,CV_8U);
   bitwise_and( the_gradient, zero_crossings, modulated_zero_crossings ); // David Vernon: put the zero-crossings modulated by image gradient in

   Mat laplacian_gray = convert_32bit_image_for_display( laplacian, 128.0 );
/***************************************************************************************************/
   imshow(log_window_name, laplacian_gray);
   imshow(zc_window_name, zero_crossings);
   imshow(mod_zc_window_name, modulated_zero_crossings);
 }
```

Demos

The following code is taken from the cannyEdgeDetection project in the lectures directory of the ACV repository

See:

```
cannyEdgeDetection.h
cannyEdgeDetectionImplementation.cpp
cannyEdgeDetectionApplication.cpp
```

```
/*
 * CannyThreshold
 * Trackbar callback - Canny thresholds input with a ratio 1:3
 */

void CannyThreshold(int, void*)
{
   extern Mat src;
   extern Mat src_gray;
   extern Mat src_blur;
   extern Mat detected_edges;
   extern int cannyThreshold;
   extern char* canny_window_name;
   extern int gaussian_std_dev;

   int ratio = 3;
   int kernel_size = 3;
   int filter_size;

   filter_size = gaussian_std_dev * 4 + 1;  // multiplier must be even to ensure an odd filter size as required by OpenCV
                                            // this places an upper limit on gaussian_std_dev of 7 to ensure the filter size < 31
                                            // which is the maximum size for the Laplacian operator

   cvtColor(src, src_gray, CV_BGR2GRAY);

   GaussianBlur(src_gray, src_blur, Size(filter_size,filter_size), gaussian_std_dev);

   Canny( src_blur, detected_edges, cannyThreshold, cannyThreshold*ratio, kernel_size );

   imshow( canny_window_name, detected_edges );
 }
```

Demos

The following code is taken from the contourExtraction project in the lectures directory of the ACV repository

See:

```
contourExtraction.h
contourExtractionImplementation.cpp
contourExtractionApplication.cpp
```

```
/*
 * ContourExtraction
 * Trackbar callback - Canny hysteresis thresholds input with a ratio 1:3 and Gaussian standard deviation
 */

void ContourExtraction(int, void*) {
   extern Mat src;
   extern Mat src_gray;
   extern Mat src_blur;
   extern Mat detected_edges;
   extern int cannyThreshold;
   extern char* canny_window_name;
   extern char* contour_window_name;
   extern int gaussian_std_dev;

   bool debug = true;
   int ratio = 3;
   int kernel_size = 3;
   int filter_size;
   vector<vector<Point>> contours;
   vector<Vec4i> hierarchy;
   Mat thresholdedImage;

   filter_size = gaussian_std_dev * 4 + 1;  // multiplier must be even to ensure an odd filter size as required by OpenCV
                                            // this places an upper limit on gaussian_std_dev of 7 to ensure the filter size < 31
                                            // which is the maximum size for the Laplacian operator
   cvtColor(src, src_gray, CV_BGR2GRAY);

   GaussianBlur(src_gray, src_blur, Size(filter_size,filter_size), gaussian_std_dev);

   Canny( src_blur, detected_edges, cannyThreshold, cannyThreshold*ratio, kernel_size );
```

```
   Mat canny_edge_image_copy = detected_edges.clone();   // clone the edge image because findContours overwrites it

   /* see http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours */
   /* and http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html         */
   findContours(canny_edge_image_copy,contours,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_NONE);

   Mat contours_image = Mat::zeros(src.size(), CV_8UC3);       // draw the contours on a black background

   for (int contour_number=0; (contour_number<(int)contours.size()); contour_number++) {
      Scalar colour( rand()&0xFF, rand()&0xFF, rand()&0xFF );  // use a random colour for each contour
      drawContours( contours_image, contours, contour_number, colour, 1, 8, hierarchy );
   }

   if (debug) printf("Number of contours %d: \n", contours.size());

   imshow( canny_window_name, detected_edges );
   imshow( contour_window_name, contours_image );
}
```

Exercises

1. Read OpenCV documentation for all OpenCV functions in sample code

2. Study utility functions in sample code

3. Replace Canny edge detection with Laplacian of Gaussian edge detection in contourExtraction. Comment on any differences you see

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

Section 4.2 Edges