Edge and local feature detection

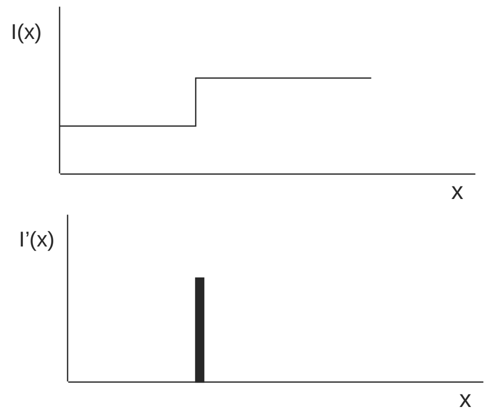
- Gradient based edge detection
- Edge detection by function fitting
- Second derivative edge detectors
- Edge linking and the construction of the chain graph

Importance of edge detection in computer vision

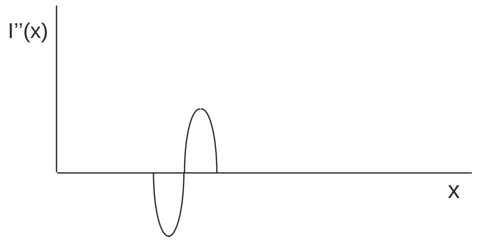
- Information reduction
  - replace image by a cartoon in which objects and surface markings are outlined
  - these are the most informative parts of the image
- Biological plausibility
  - initial stages of mammalian vision systems involve detection of edges and local features

1-D edge detection

➤ An ideal edge is a step function



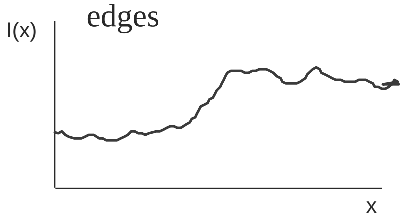
1-D edge detection

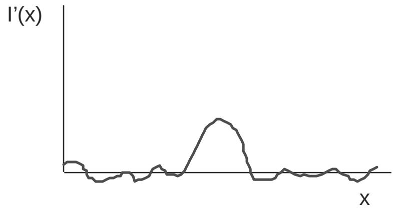


➤ The first derivative of I(x) has a **peak** at the edge
➤ The second derivative of I(x) has a **zero crossing** at the edge

1-D edge detection

- More realistically, image edges are **blurred** and the regions that meet at those edges have **noise** or variations in intensity.
  - blur - high first derivatives near edges
  - noise - high first derivatives within regions that meet at edges





Edge detection in 2-D

- Let f(x,y) be the image intensity function. It has derivatives in all directions
  - the **gradient** is a vector whose first component is the direction in which the first derivative is highest, and whose second component is the magnitude of the first derivative in that direction.
- If f is continuous and differentiable, then its gradient can be determined from the directional derivatives in any two orthogonal directions - standard to use x and y
  - magnitude = $[(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2]^{1/2}$
  - direction = $\tan^{-1}(\frac{\partial f / \partial y}{\partial f / \partial x})$

Edge detection in 2-D

- With a digital image, the partial derivatives are replaced by finite differences:
  - $\Delta_x f = f(x,y) - f(x-1, y)$
  - $\Delta_y f = f(x,y) - f(x, y-1)$
- Alternatives are:
  - $\Delta_{2x} f = f(x+1,y) - f(x-1,y)$
  - $\Delta_{2y} f = f(x,y+1) - f(x,y-1)$
- Robert’s gradient
  - $\Delta_+ f = f(x+1,y+1) - f(x,y)$ $\nearrow \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}$
  - $\Delta_- f = f(x,y+1) - f(x+1, y)$ $\searrow \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$

Edge detection in 2-D

- How do we combine the directional derivatives to compute the gradient magnitude?
  - use the root mean square (RMS) as in the continuous case
  - take the maximum absolute value of the directional derivatives

Combining smoothing and differentiation - fixed scale

- Local operators like the Roberts give high responses to any intensity variation
  - local surface texture
- If the picture is first smoothed by an averaging process, then these local variations are removed and what remains are the “prominent” edges
  - smoothing is blurring, and details are removed
- Example $f_{2x2}(x,y) = 1/4[f(x,y) + f(x+1,y) + f(x,y+1) + f(x+1,y+1)]$

Smoothing - basic problems

- What function should be used to smooth or average the image before differentiation?
  - box filters or uniform smoothing
    - easy to compute
    - for large smoothing neighborhoods assigns too much weight to points far from an edge
  - Gaussian, or exponential, smoothing

$$(1/2\pi\sigma)e^{-(x^2+y^2)/2\sigma^2}$$

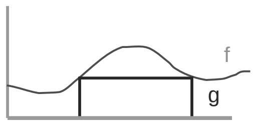
Smoothing and convolution

- The convolution of two functions, f(x) and g(x) is defined as

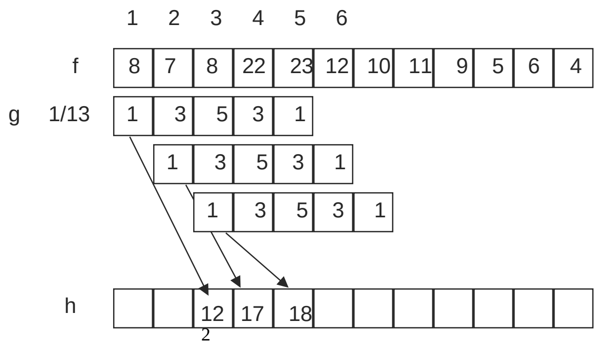
$$h(x) = \int_{-\infty}^{\infty} g(x')f(x-x')dx' = g(x) * f(x)$$

- When the functions f and g are discrete and when g is nonzero only over a finite range [-n,n] then this integral is replaced by the following summation:

$$h(i) = \sum_{j=-n}^{n} g(j)f(i+j)$$



Example of 1-d convolution



$$h(4) = \sum_{j=-2}^{2} g(j)f(4+j)$$

$$= g(-2)f(2) + g(-1)f(3) + g(0)f(4) + g(1)f(5) + g(2)f(6)$$

Smoothing and convolution

- These integrals and summations extend simply to functions of two variables:

$$h(i,j) = f(i,j) * g = \sum_{k=-n}^{n} \sum_{l=-n}^{n} g(k,l) f(i+k, j+l)$$

- Convolution computes the weighted sum of the gray levels in each nxn neighborhood of the image, f, using the matrix of weights g.
- Convolution is a so-called linear operator because
  - $g*(af_1 + bf_2) = a(g*f_1) + b(g*f_2)$

2-D convolution

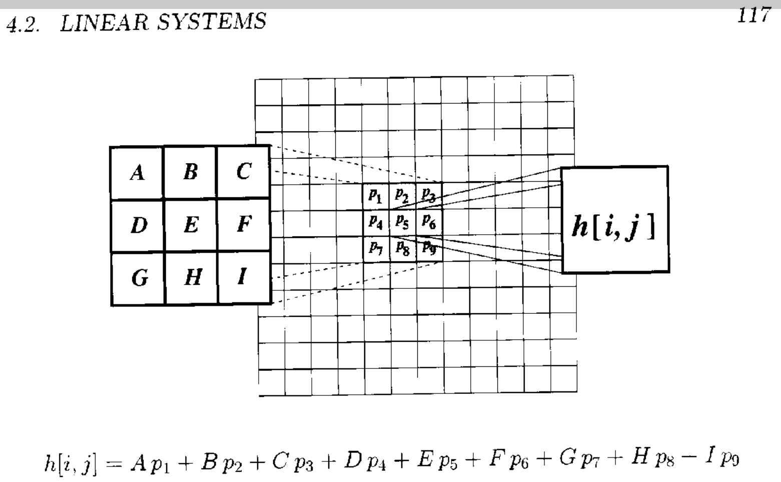
$$h(5,5) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} g(k,l) f(5+k, 5+l)$$

$$= g(-1,-1) f(4,4) + g(-1,0) f(4,5) + g(-1,1) f(4,4)$$

$$+ g(0,-1) f(5,4) + g(0,0) f(5,5) + g(0,1) f(5,6)$$

$$+ g(1,-1) f(6,4) + g(1,0) f(6,5) + g(1,1) f(6,6)$$

Smoothing and convolution



4.2. LINEAR SYSTEMS 117

$$h[i,j] = A\,p_1 + B\,p_2 + C\,p_3 + D\,p_4 + E\,p_5 + F\,p_6 + G\,p_7 + H\,p_8 - I\,p_9$$

Gaussian smoothing

- Advantages of Gaussian filtering
  - rotationally symmetric (for large filters)
  - filter weights decrease monotonically from central peak, giving most weight to central pixels
  - Simple and intuitive relationship between size of σ and size of objects whose edges will be detected in image.
  - The gaussian is separable:

$$e^{-\frac{(x^2+y^2)}{2\sigma^2}} = e^{-\frac{x^2}{2\sigma^2}} * e^{-\frac{y^2}{2\sigma^2}}$$

Advantage of seperability

- First convolve the image with a one dimensional horizontal filter
- Then convolve the result of the first convolution with a one dimensional vertical filter
- For a kxk Gaussian filter, 2D convolution requires k^2 operations per pixel
- But using the separable filters, we reduce this to 2k operations per pixel.

Separability

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} \quad \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix} \quad \begin{bmatrix} & & \\ & 65 & \\ & & \end{bmatrix}$$

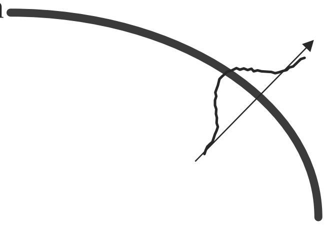
$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} \quad \begin{array}{r} =2 + 6 + 3 = 11 \\ = 6 + 20 + 10 = 36 \\ = 4 + 8 + 6 = \underline{18} \\ 65 \end{array}$$

Advantages of Gaussians

- Convolution of a Gaussian with itself is another Gaussian
  - so we can first smooth an image with a small Gaussian
  - then, we convolve that smoothed image with another small Gaussian and the result is equivalent to smoother the original image with a larger Gaussian.
  - If we smooth an image with a Gaussian having sd σ twice, then we get the same result as smoothing the image with a Gaussian having standard deviation $(2\sigma)^{1/2}$

Combining smoothing and differentiation - fixed scale

- Non-maxima supression - Retain a point as an edge point if:
  - its gradient magnitude is higher than a threshold
  - its gradient magnitude is a local maxima in the gradient direction



simple thresholding will compute thick edges

Summary of basic edge detection steps

- Smooth the image to reduce the effects of local intensity variations
  - choice of smoothing operator practically important
- Differentiate the smoothed image using a digital gradient operator that assigns a magnitude and direction of the gradient at each pixel
- Threshold the gradient magnitude to eliminate low contrast edges

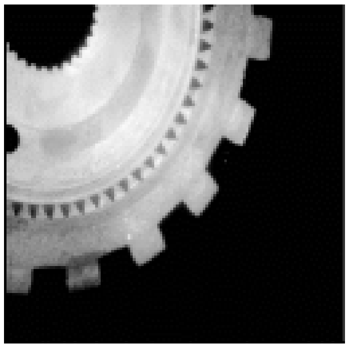
Summary of basic edge detection steps

- Apply a nonmaxima suppression step to thin the edges to single pixel wide edges
  - the smoothing will produce an image in which the contrast at an edge is spread out in the neighborhood of the edge
  - thresholding operation will produce thick edges

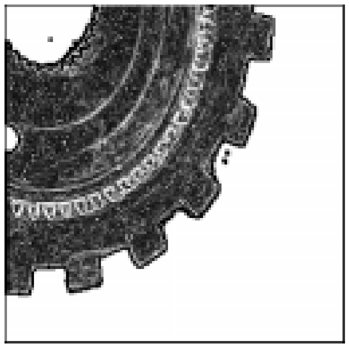
The scale-space problem

- Usually, any single choice of σ does not produce a good edge map
  - a large σ will produce edges form only the largest objects, and they will not accurately delineate the object because the smoothing reduces shape detail
  - a small σ will produce many edges and very jagged boundaries of many objects.
- Scale-space approaches
  - detect edges at a range of scales $[\sigma_1, \sigma_2]$
  - combine the resulting edge maps
    - trace edges detected using large σ down through scale space to obtain more accurate spatial localization.

Examples

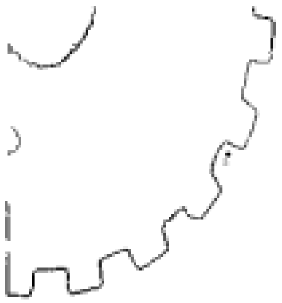


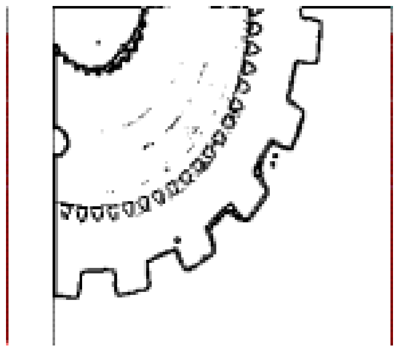
Gear image



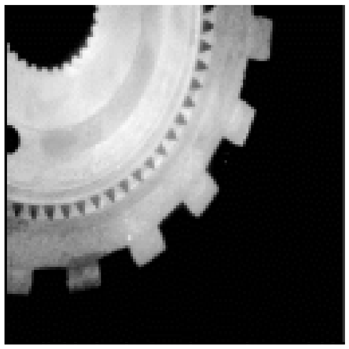
3x3 Gradient magnitude

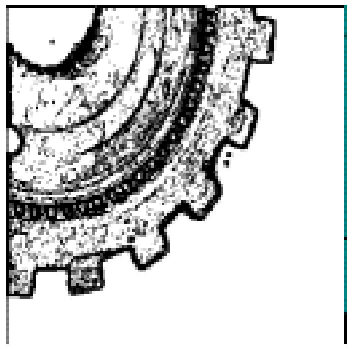
Examples


High threshold

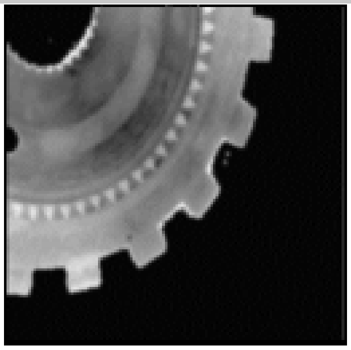

Medium threshold

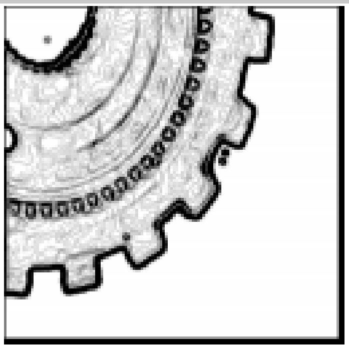
Examples



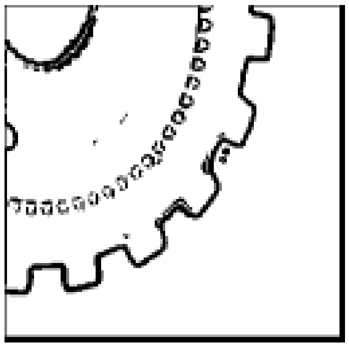

low threshold

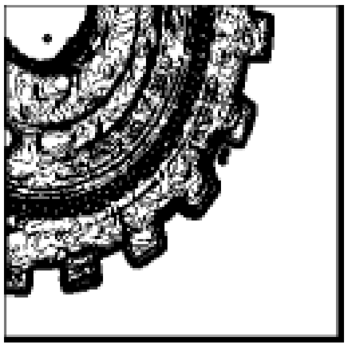
Examples


Smoothed 5x5 Gaussian

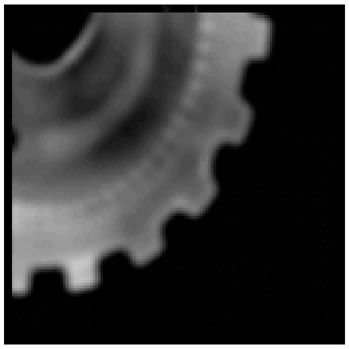

3x3 gradient magnitude

Examples

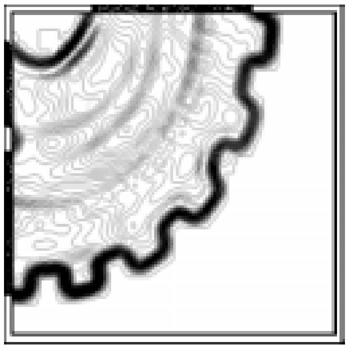




Examples

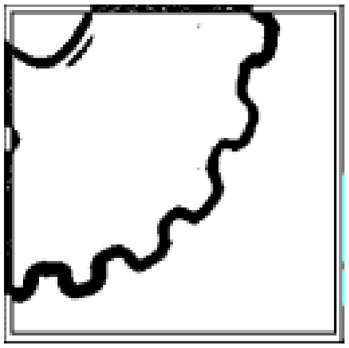


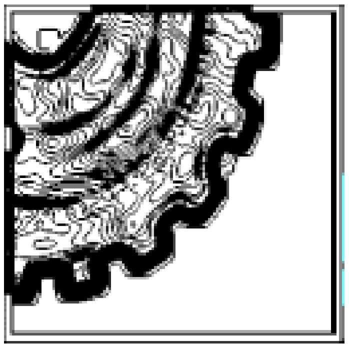
smoothed 15x15 Gaussian



3x3 gradient magnitude

Examples





Laplacian edge detectors

- Directional second derivative in direction of gradient has a zero crossing at gradient maxima
- Can "approximate" directional second derivative with Laplacian

$$\partial^2 f / \partial x^2 + \partial^2 f / \partial y^2$$

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

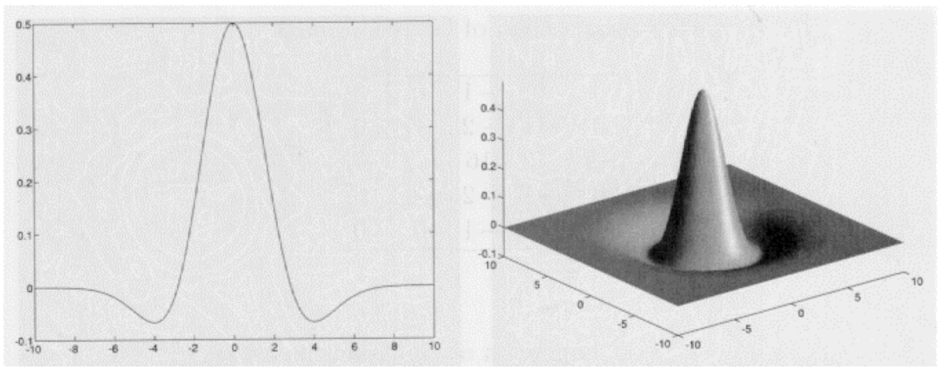
- Its digital approximation is
  - $\nabla^2 f(x,y) = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4 f(x,y)$

    $= [f(x+1,y) - f(x,y)] - [f(x,y) - f(x-1,y)] + [f(x,y+1)-f(x,y)] - [f(x,y) - f(x,y-1)]$

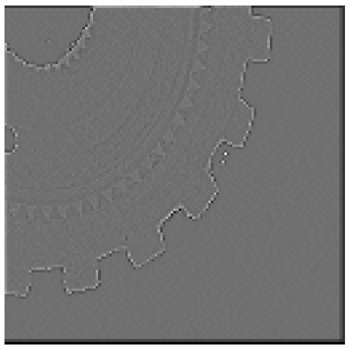
Laplacian edge detectors

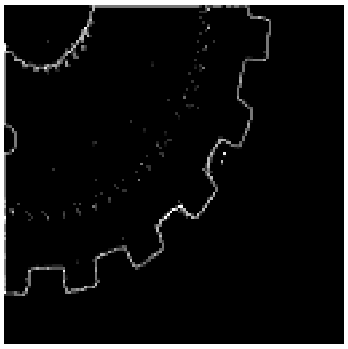
- Laplacians are also combined with smoothing for edge detectors
  - Take the Laplacian of a Gaussian smoothed image - called the Mexican Hat operator or DoG (Difference of Gaussians)
  - Locate the zero-crossing of the operator
    - these are pixels whose DoG is positive and which have neighbor's whose DoG is negative or zero
  - Usually, measure the gradient or directional first derivatives at these points to eliminate low contrast edges.

Laplacian of Gaussian or “Mexican Hat”

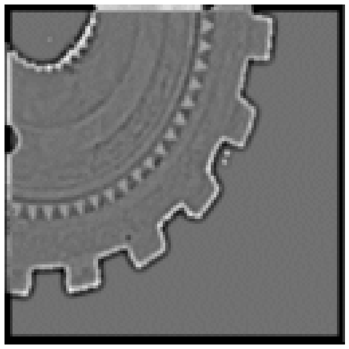


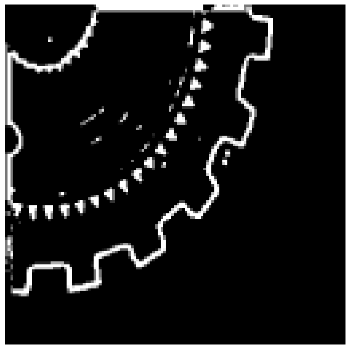
Laplacian of Gaussian


5x5 Mexican Hat - Laplacian of Gaussian


Zero crossings

Laplacian of Gaussian


13 x 13 Mexican hat


zero crossings

Edge linking and following

- Group edge pixels into chains and chains into large pieces of object boundary.
  - can use the shapes of long edge chains in recognition
    - slopes
    - curvature
    - corners

Edge linking and following

- Basic steps
  - thin connected components of edges to one pixel thick
  - find simply connected paths
  - link them at corners into a graph model of image contours
    - optionally introduce additional corners on interiors of simple paths
  - compute local and global properties of contours and corners

Thinning

- Consider a 3x3 neighborhood of a binary image in which the center pixel is “1”
  - the center point is a simple point if changing it from a 1 to a 0 does not change the number of connected component of the 3x3 neighborhood.

$$\begin{array}{ccc} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{array} \qquad \begin{array}{ccc} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{array}$$

  - the first is 8-simple but not 4-simple
  - the second is neither 4 nor 8 simple

Thinning

- Removal of a simple point will not change the number of connected components in a binary image
- An end point is a 1 with exactly one 1-neighbor

Thinning

- A 1-pixel (i,j) in a binary image is a North border point if pixel (i,j+1) is a 0.
  - similarly define East, West and South border points.
- Simple thinning algorithm
  - For D = N,E,W,S do

    Eliminate all D border points that are simple points and NOT end points
- Must do the directions in sequence and not together or we could erase a component
- Result depends on the order in which the directions are considered

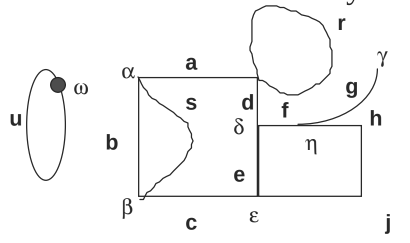
```
0 0 0 0 0 0 0
0 1 1 1 1 0 0
0 1 1 1 1 0 0
0 0 0 0 0 0 0
```

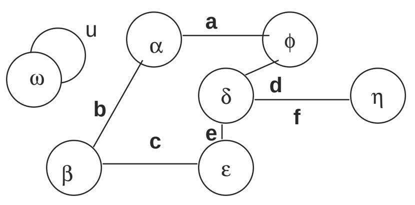
Example - 4 simple points

```
0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0
0 1 0 1 1 1 0 0          0 1 0 0 1 1 0 0          0 1 0 0 1 1 0 0
0 1 0 1 1 0 0 1    N     0 1 0 1 1 0 0 1    E     0 1 0 1 1 0 0 1
0 1 1 1 1 1 1 1  ----->  0 1 0 1 1 0 1 1  ----->  0 1 0 1 0 0 1 1
0 1 1 1 1 1 1 0          0 1 1 1 1 1 1 0          0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0          0 1 1 0 0 1 1 0          0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0
                                |
                              W |
                                v
                         0 0 0 0 0 0 0 0
                         0 1 0 0 1 1 0 0
                         0 1 0 0 1 0 0 1
                         0 1 0 0 1 0 1 1
                         0 1 1 1 1 1 1 0
                         0 0 1 0 0 0 1 0
                         0 0 0 0 0 0 0 0
```

Finding simply connected chains

- Goal: create a graph structured representation (chain graph) of the image contours
  - vertex for each junction in the image
  - edge connecting vertices corresponding to junctions that are connected by a chain; edge labeled with chain





Partial graph

Creating the chain graph

- Algorithm: given binary image, E, of thinned edges
  - create a binary image, J, of junctions and end points
    - points in E that are 1 and have more than two neighbors that are 1 or exactly one neighbor that is a 1
  - create the image E-J = C(chains)
    - this image contains the chains of E, but they are broken at junctions

Creating the chain graph

- Perform a connected component analysis of C. For each component store in a table T:
  - its end points (0 or 2)
  - the list of coordinates joining its end points
- For each point in J:
  - create a node in the chain graph , G, with a unique label

Creating the chain graph

- For each chain in C
  - if that chain is a closed loop (has no end points)
    - choose one point from the chain randomly and create a new node in G corresponding to that point
    - mark that point as a “loop junction” to distinguish it from other junctions
    - create an edge in G connecting this new node to itself, and mark that edge with the name of the chain loop
  - if that chain is not a closed loop, then it has two end points
    - create an edge in G linking the two points from J adjacent to its end points

Creating the chain graph

- Data structure for creating the chain graph
- Biggest problem is determining for each open chain in C the points in J that are adjacent to its end points
  - create image J in which all 1’s are marked with their unique labels.
  - For each chain in C
    - Examine the 3x3 neighborhood of each end point of C in J
    - Find the name of the junction or end point adjacent to that end point from this 3x3 neighborhood.

Finding internal “corners” of chains

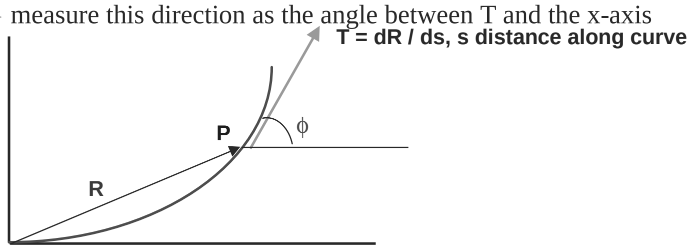
- Chains are only broken at junctions
  - but important features of the chain might occur at internal points
  - example: closed loop corresponding to a square - would like to find the natural corners of the square and add them as junctions to the chain graph (splitting the chains at those natural corners)
- Curve segmentation
  - similar to image segmentation, but in a 1-D form
    - local methods, like edge detectors
    - global methods, like region analyzers

Local methods of curve segmentation

- Natural locations to segment contours are points where the slope of the curve is changing quickly
  - these correspond, perceptually, to “corners” of the curve.
- To measure the change in slope we are measuring the curvature of the curve
  - straight line has 0 curvature
  - circular arc has constant curvature corresponding to 1/r
  - Can estimate curvature by fitting a simple function (circular arc, quadratic function, cubic function) to each neighborhood of a chain, and using the parameters of the fit to estimate the curvature at the center of the neighborhood.

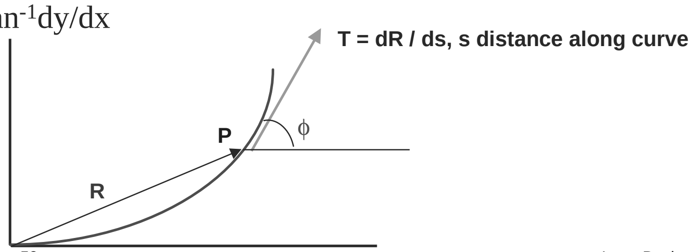
Formulae for curvature

- Consider moving a point, P, along a curve.
  - Let T be the unit tangent vector as P moves
    - T has constant length (1)
    - but the direction of T, ϕ, changes from point to point unless the curve is a straight line
    - measure this direction as the angle between T and the x-axis



Formulae for curvature

- The curvature, κ, is the instantaneous rate of change of ϕ with respect to s, distance along the curve
  - $\kappa = d\phi / ds$
  - $ds = [dx^2 + dy^2]^{1/2}$
  - $\phi = \tan^{-1} dy/dx$



Formulae for curvature

- Now

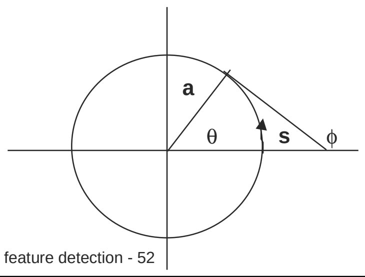
$$d\phi / dx = \frac{\frac{d^2y}{dx^2}}{1+(\frac{dy}{dx})^2}$$

and $$ds / dx = \sqrt{1+(\frac{dy}{dx})^2}$$

so $$\kappa = d\phi / ds = \frac{d\phi / dx}{ds / dx} = \frac{d^2y/dx^2}{[1+(\frac{dy}{dx})^2]^{3/2}}$$

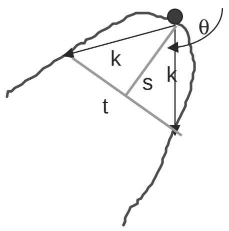
Example - circle

- For the circle
  - s = aθ
  - ϕ = θ + π/2
  - so κ = dϕ/ds = dθ/adθ = 1/a



Local methods of curve segmentation

- There are also a wide variety of heuristic methods to estimate curvature-like local properties
  - For each point, p , along the curve
  - Find the points k pixels before and after p on the curve (p^{+k}, p^{-k}) and then measure
    - the angle between pp^{+k} and pp^{-k}
    - the ratio s/t



Local methods of curve segmentation

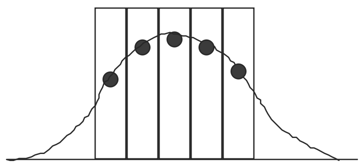
- Similar problems to edge detection
  - what is the appropriate size for k?
  - how do we combine the curvature estimates at different scales?
  - boundary problems near the ends of open curves - not enough pixels to look out k in both directions

Back to smoothing functions

- To smooth an image using a Gaussian filter we must
  - choose an appropriate value for σ, which controls how quickly the Gaussian falls to near zero
    - small σ produces filter which drops to near zero quickly - can be implemented using small digital array of weights
    - large σ produces a filter which drops to near zero slowly - will be implemented using a larger size digital array of weights
  - determine the size weight array needed to adequately represent that Gaussian
    - choose a size for which the values at the edges of the weight array are 10^{-k} as large as the center weight
    - weight array needs to be of odd size to allow for symmetry

Gaussian smoothing

- To smooth an image using a Gaussian filter we must
  - sample the Gaussian by integrating it over the square pixels of the array of weights and multiplying by the scale factor to obtain integer weights



Gaussian smoothing

- Because we have truncated the Gaussian the weights will not sum to 1.0 x scale factor
  - in “flat” areas of the image we expect our smoothing filter to leave the image unchanged
  - but if the filter weights do not sum to 1.0 x scale factor, it will either amplify (> 1.0) or de-amplify the image
  - normalize the weight array by dividing each entry by the sum of the all of the entries
  - convert to integers

Edge detection by function fitting

- General approach
  - fit a function to each neighborhood of the image
  - use the gradient of the function as the digital gradient of the image neighborhood

Edge detection by function fitting

- Example: fit a plane to a 2x2 neighborhood
  - z = ax + by + c; z is gray level - need to determine a,b,c
  - gradient is then $(a^2 + b^2)^{1/2}$
  - neighborhood points are f(x,y), f(x+1,y), f(x,y+1) and f(x+1,y+1)
- Need to minimize

$$E(a,b,c) = \sum_{i=0}^{1}\sum_{j=0}^{1}[a(x+i) + b(y+j) + c - f(x+i,y+j)]^2$$

- Solve this and similar problems by:
  - differentiating with respect to a,b,c, setting results to 0, and
  - solving for a,b,c in resulting system of equations

Edge detection by function fitting

- E/ a = ΣΣ2[a(x+i) + b(y+j) + c - f(x+i,y+j)](x+i)
- E/ b = ΣΣ2[a(x+i) + b(y+j) + c - f(x+i,y+j)](y+j)
- E/ c = ΣΣ2[a(x+i) + b(y+j) + c - f(x+i,y+j)]
- It is easy to verify that

  a = [f(x+1,y) + f(x+1,y+1) - f(x,y) - f(x,y+1)]/2

  b = [f(x,y+1) + f(x+1,y+1) - f(x,y) - f(x+1,y)]/2
- a and b are the x and y partial derivatives

$$a = \begin{array}{cc} -1 & 1 \\ -1 & 1 \end{array} \qquad b = \begin{array}{cc} 1 & 1 \\ -1 & -1 \end{array}$$

Edge detection by function fitting

- Could also fit a higher order surface than a plane
  - with a second order surface we could find the (linear) combination of pixel values that corresponds to the higher order derivatives, which can also be used for edge detection
- Would ordinarily use a neighborhood larger than 2x2
  - better fit
  - for high degree functions need more points for the fit to be reliable.