1. **Histograms**

- Texture is made up of repeated local patterns, so to represent texture we need to:

+ Use filters that look like patterns and consider magnitude of response to find the patterns.

+ Use mean, standard deviation, histogram and histogram of “prototypical” feature occurrences to describe their statistics.

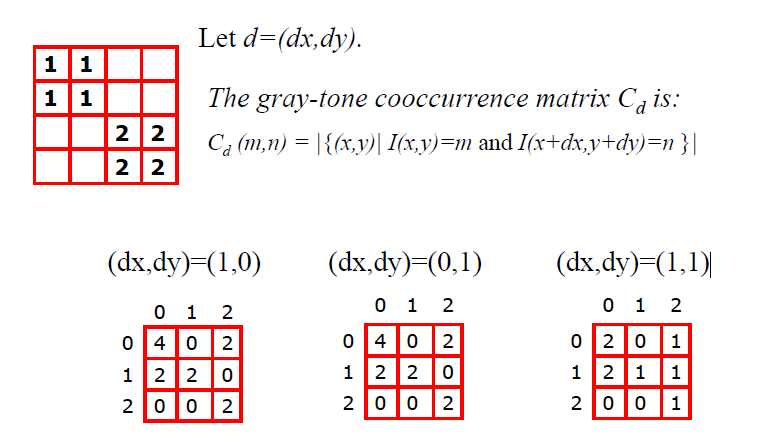
*Comparing histograms is the simplest texture discrimination.*

- Dividing intensities into discrete ranges and counting how many pixels in each range are the steps to compare histograms. One may calculate chi square distance between texton histograms. Distance reveals dissimilar texture.

*Texture representation:*

- Original image is filtered, squared and statistics to summarize patterns in small windows. Mean d/dx and d/dy values in statistics table are used to group consistent pixels in windows with primarily horizontal and vertical edges. Finally, compare dissimilarity texture by computing distance between pixels in groups.

1. **Co-occurrence Matrices and Features**

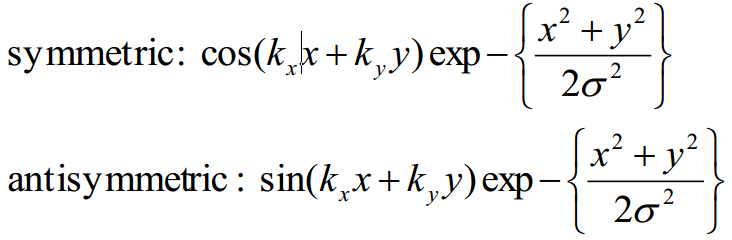


1. **Filter banks**

* Before, we used two filters, and result in a 2-dimensional feature vector to describe texture in a window. Besides, we can generalize to apply a collection of multiple (d) filters: a “filter bank”, and result in d-dimensional feature vectors.
* Filters, which are combination of scales and orientations, different types of patterns, will be put in the bank.

1. **Gabor Filters**

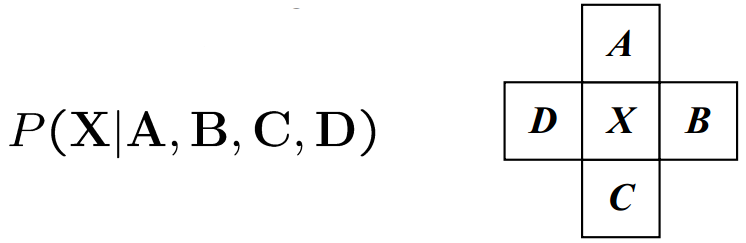
* Gabor filters at different scales and spatial frequencies.
* Its impulse response is defined by a sinusoidal wave (a plane wave for 2D Gabor filters) multiplied by a Gaussian function.
* Top row shows symmetric (or even) filters and bottom row shows the anti-symmetric (or odd) filters:



1. **Markov Random Field**

A Markov random field (MRF) is generalization of Markov chains to two or more dimensions.

First-order MRF: probability that pixel X takes a certain value given the values of neighbors A, B, C, and D:



1. **Markov Chain – Transition Table**

Markov Chain

•a sequence of random variables x1,… , xn

• is the state of the model at time t: xt

1. **Texture synthesis:**

***Goal: create new samples of a given texture***

***Most basic algorithm***

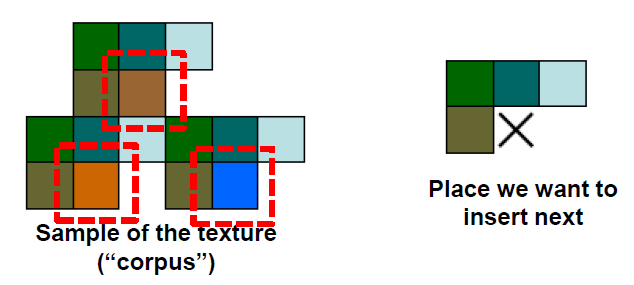
Build probability histogram

* find all blocks of N consecutive words/letters in training documents
* compute probability of occurrence

***Texture synthesis: intuition***

Before, we inserted the next word based on existing nearby words…

Now we want to insert pixel intensities based on existing nearby pixel values.



***Synthesizing One Pixel***

- What is P (x| neighborhood of pixels around x)?

- Find all the windows in the image that match the neighborhood

- To synthesize x

+ pick one matching window at random

+ assign x to be the center pixel of that window

An exact neighborhood match might not be present, so find the best matches using SSD error and randomly choose between them, preferring better matches with higher probability

1. **Unit of synthesis – block**

Image Quilting

Observation: neighbor pixels are highly correlated

Idea: unit of synthesis = block

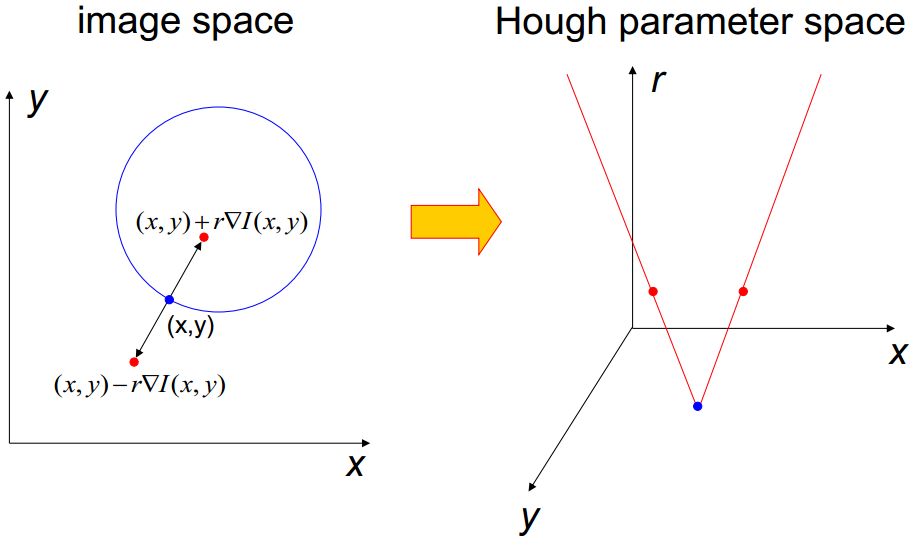
Exactly the same but now we want P (B|N(B))

Much faster: synthesize all pixels in a block at once

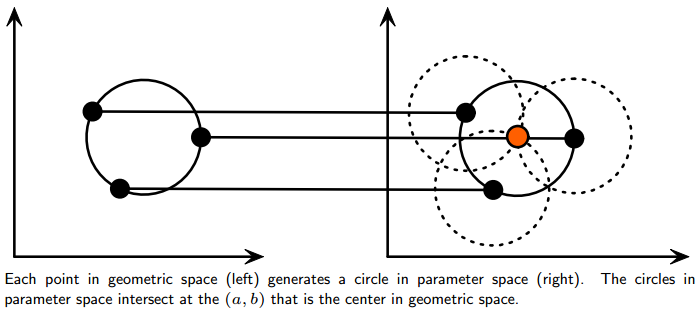
Input image with block form are performed random placement of blocks. Neighboring blocks constrained by overlap and then, minimal error boundary cut

1. **How to detect circle?**

* The parameter space will have 3 dimensions (x,y,r)
* For each (x,y,r), draw the corresponding circle in the image and compute its “support”



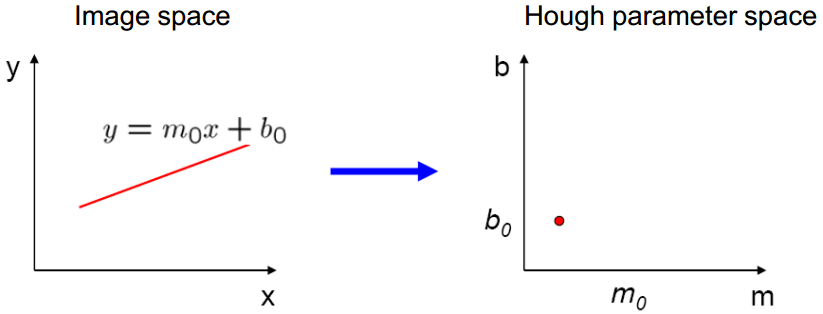
* If the circles in an image are of known radius R, then the search can be reduced to 2D. The locus of (a, b) points in the parameter space fall on a circle of radius R centered at (x, y). The true center point will be common to all parameter circles, and can be found with a Hough accumulation array.



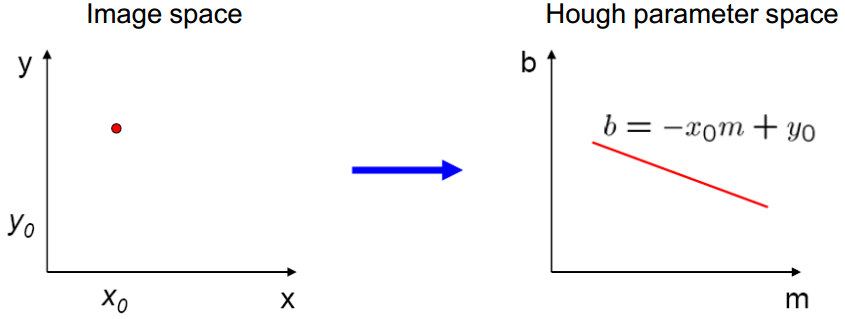
1. **How to detect line?**

**\* In (m,b) space:**

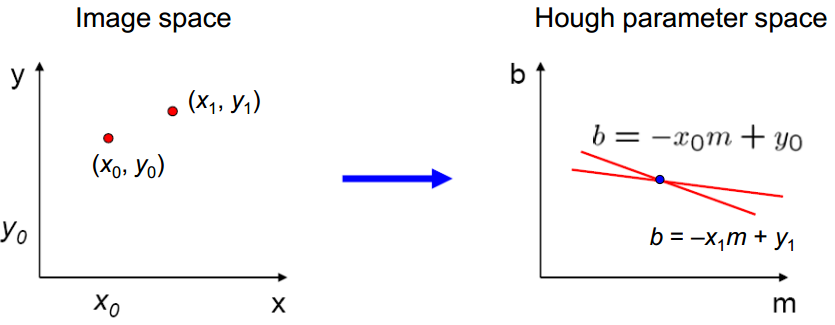
* A line in the image corresponds to a point in Hough space



* A point in the image corresponds to a line in Hough space



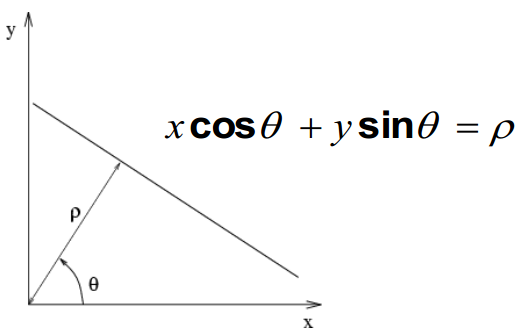
* A line that contains both (*x0*, *y0*) and (*x1*, *y1*) is the intersection of the lines *b* = -*x0m* + *y0* and *b* = *-x1m* + *y1*.



=> Problem:

* Unbounded parameter domains
* Vertical lines require infinite m

**\* Alternative:** polar representation



Each point will add a sinusoid in the (*θ*, *ρ*) parameter space

**Algorithm outline**

* Initialize accumulator H to all zeros
* For each edge point (*x,y*) in the image

For *θ* = 0 to 180

ρ = *x* cos *θ* + *y* sin *θ*

H(*θ*, *ρ*) = H(*θ*, *ρ*) + 1

end

end

* Find the value(s) of (*θ*, *ρ*) where H(*θ*, *ρ*) is a local maximum
* The detected line in the image is given by ρ = *x* cos *θ* + y sin *θ*

**Incorporating image gradients**

* Recall: when we detect an edge point, we also know its gradient direction
* But this means that the line is uniquely determined!
* Modified Hough transform:

For each edge point (*x*, *y*)

*θ* = gradient orientation at (*x*, *y*)

*ρ* = *x* cos *θ* + *y* sin *θ*

H(*θ*, *ρ*) = H(*θ*, *ρ*) + 1

end



1. **Prove :**  *(Mask Operation Edge Detection - Properties of Gradient)*

* The gradients of I(x, y) are Ix(x, y) and Iy(x, y)
* For a new position (x + ∆ cos θ, y + ∆ sin θ), by Tayler’s expansion, we have

*I*(*x +* ∆ cos *θ*, *y +* ∆ sin *θ*) ≅ *I*(*x*, *y*) + ∆ cos *θ Ix*(*x*, *y*) + ∆ sin *θ Iy*(*x*, *y*)

* Set v ≡ (∆ cos θ, ∆ sin θ) and ∇I = (Ix, Iy).

⇒ *I*(*x +* ∆ cos *θ*, *y +* ∆ sin *θ*) - *I*(*x*, *y*) ≅ <*v,* ∇I> = |*v* ||∇I| cos *α*

*α* : the angle between *v and* ∇I

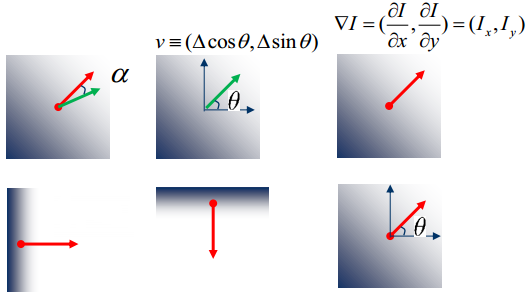
*α* = 0 ⇒ <*v,* ∇I> is maximized ⇒ *v* and ∇I are the same orientation

⇒ ∇I is ***vertical*** to edge orientation

*Iy* = 0 ⇒ ∇I = (*Ix*, 0) ⇒ Orientation of ∇I is ***horizontal***

*Ix* = 0 ⇒ ∇I = (0, *Ix*) ⇒ Orientation of ∇I is **vertic*al***

*Ix* ≠ 0 ⇒



1. **Derivation theorem of convolution**

Consider a single row or column of the image.

Plotting intensity as a function of position gives a signal.

To find edges, look for peaks in

We need 4 operations: signal *f*, kernel *g*, convolution (*f \* g*), and differentiations .

Differentiation is convolution, and convolution is associative .

This saves us one operation, and we only need 3 operations: signal *f*, kernel , and convolution .

1. **Mask for Gaussian Function**

Mask: , where *σ* = 1~10

When *σ* = 1, the mask becomes .

Gaussian Mask 5x5, where *i*, *j* = {-2, -1, 0, 1, 2}

****

Integer mask will be better for computation.

Choose the minimum of h[*i*, *j*] to normalize

Integer mask for Gaussian function

****

Sum of the weights should be 1.

Normalization by

1. **Sharpening**

Sharpening: to enhance line structures or other details in an image.

Sharpening filter: accentuates differences with local average (Note that filter sums to 1)

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

−