

SECTION 5 IMAGE PROCESSING	2
5.1 Resampling	3
5.1.1 Image Interpolation Comparison	3
5.2 Convolution	3
5.3 Smoothing Filters	3
5.3.1 Mean Filter	3
5.3.2 Median Filter	4
5.3.3 Pseudomedian Filter	6
5.3.4 Gaussian Smoothing	6
5.4 Edge Detectors	7
5.4.1 Roberts Cross Edge Detector	7
5.4.2 Sobel Edge Detector	8
5.4.3 Prewitt / Compass Edge Detector	9
5.4.4 Quadratic Edge Detector	9
5.4.5 Canny Edge Detector	9
5.5 Boolean Mask Operators	9
5.5.1 Boolean AND	9
5.5.2 Boolean OR	10
5.5.3 Boolean XOR	10
5.5.4 Boolean NOT	11
5.5.5 Boolean NAND	11
5.5.6 Boolean NOR	12
5.5.7 Boolean XNOR	12
5.6 Morphological Operators	13
5.6.1 Erosion	13
5.6.2 Dilation	13
5.6.3 Opening	14
5.6.4 Closing	15
5.6.5 Thinning	15
5.6.6 Thickening	16
5.6.7 Hit or Miss Transform	17
5.6.8 Skeletonization	17
5.6.9 Pruning	18
5.6.10 Perimeter Detection	18
5.7 Image Enhancement Techniques	18
5.7.1 Linear Contrast Stretching	18
5.7.2 Piecewise Contrast Stretch	18
5.7.3 Histogram Equalization	18
5.7.4 Unsharp Masking	19
5.7.5 Binary Thresholding	19
5.7.6 Color Balancing	19
5.7.7 Gamma Correction	19
5.8 Texture Growth	19
5.9 Hough Transforms	19
5.10 Segmentation	19
5.11 Wavelets	19
5.12 References	19

SECTION 5 IMAGE PROCESSING

5.1 Resampling

[TBD]

5.1.1 Image Interpolation Comparison

[TBD]

5.2 Convolution

[TBD]

5.3 Smoothing Filters

5.3.1 Mean Filter

Description: A mean filter is the simplest type of convolution smoothing. The mean filter takes an average of the pixel values over the convolution kernel and places the value in the new image. This is done by making the kernel sum to 1 to conserve energy. The effect is used to blur an image and reduce pixel noise.

To qualify as a mean filter all the elements in the kernel must be equal to each other or equal to zero. The kernel size and shape affects the appearance of the blur. The most typical is a square kernel:

$$\frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This evenly blurs the pixel content from the 9 surrounding pixels into the new pixel. The kernel does not have to be square though. Rectangular kernels can be used as well:

$$\frac{1}{10} \cdot [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$$

A rectangular component will give the smoothing a motion blurred effect. The kernel can also include zeros as elements, as long as the overall sum of the kernel is 1 and all the elements that are not zero have the same value.

$$\frac{1}{10} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This kernel will give the smoothing a diagonal appearance. Of course, there are many other shapes that will give alternative results, but these few are the most relevant. The result from each of the three preceding kernels is shown below.

Note: the last two examples above use larger kernel sizes to magnify the effect of the blur to make it more apparent in the examples below.



Original



Blurred with Square Mean Kernel



Blurred with Horizontal Kernel



Blurred with Diagonal Kernel

Some other interesting effect can be achieved with less conventional kernels. For instance, using a horizontal kernel with only the ends populated gives the input image the appearance of double vision.

$$\frac{1}{2} \cdot [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]$$



5.3.2 Median Filter

Description: Median filtering is a pseudo-convolution operation that makes an image out of the median values of neighborhoods of pixels in the original image. This filter is used to reduce noise in an image and tends to be less destructive to the original image than the mean filter.

A 3x3 median filter has the form of:

$$\tilde{P}_{i,j} = \text{median} \left(\begin{bmatrix} P_{i-1,j-1} & P_{i-1,j} & P_{i-1,j+1} \\ P_{i,j-1} & P_{i,j} & P_{i,j+1} \\ P_{i+1,j-1} & P_{i+1,j} & P_{i+1,j+1} \end{bmatrix} \right)$$

Where $P_{i,j}$ is the current pixel and the rest represent the neighborhood around it. The median is calculated and returned to the median image at point $\tilde{P}_{i,j}$. The median for an odd sized kernel (total number of elements is odd) is simply the middle number when the kernel values are sorted. For an even sized kernel the median is defined as the average of the two centermost values when the kernel values are sorted.

$$\text{median} \left(\begin{bmatrix} 4 & 6 & 11 \\ 2 & 3 & 9 \\ 15 & 8 & 7 \end{bmatrix} \right) = 5$$

$$\text{median} \left(\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} \right) = 2.5$$

The result of 5x5 and 11x11 median filters are shown below:



Original



Blurred with 5x5 Median Filter



Blurred with 11x11 Median Filter

The filter effectively smooths the minor variations in the scene and leaves the color and edges relatively untouched, because the median filter doesn't act like a low-pass filter in the way the mean filter does. The median filter is able to retain meaningful high frequency edges while reducing general noise. The figure below shows the difference between a 5x5 median filter and a 5x5 mean filter on the same image.



Blurred with 5x5 Median Filter



Blurred with 5x5 Mean Filter

The edges in the mean filtered image are badly degraded while the edges in the median filtered image are still crisp.

5.3.3 Pseudomedian Filter

[TBD]

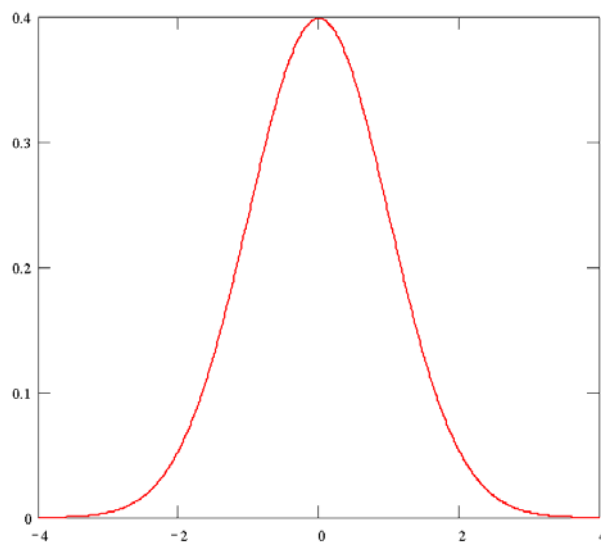
5.3.4 Gaussian Smoothing

Description: Smooths an image using a weighted mean based on the 2D Gaussian point spread function.

The 2D general form of the Gaussian equation

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

Where x is the independent variable, μ is the mean, and σ is the standard deviation of the set. The shape of the Gaussian with a mean of 0 and a standard deviation of 1 is shown below.



The circularly symmetric 3D Gaussian equation centered at the origin has the form of:

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

[TBD]



5.4 Edge Detectors

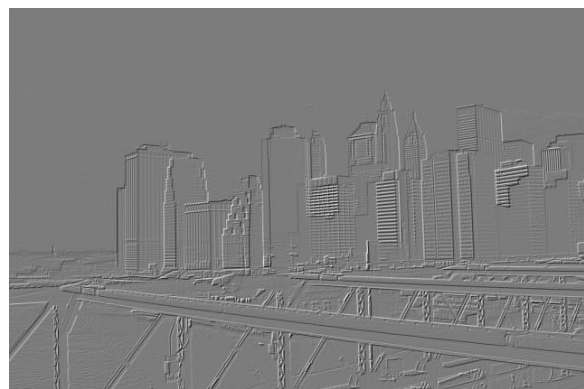
5.4.1 Roberts Cross Edge Detector

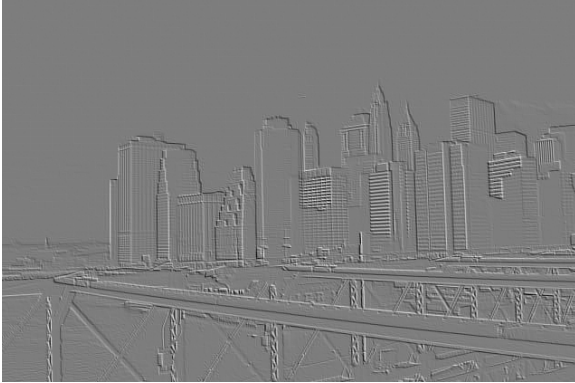
Description: The Robert's Cross edge detector is one of the simplest convolution operations that can be run on a digital image. Robert's Cross is a simple edge detector that consists of two 2x2 kernels run over an image to find gradient edges.

The two kernels are:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Convolving the image with the first kernel finds the gradient edges at -45° and the second at 45° . The images below show a test image and the resulting edges. The top-left image is the original, the top-right is the result of the first convolution, and the bottom-left is the result of the second convolution.





The bottom-right image shows the results from taking the root-sum-square (RSS) of the two images. This gives an image that has values that increase with the strength of the edge.

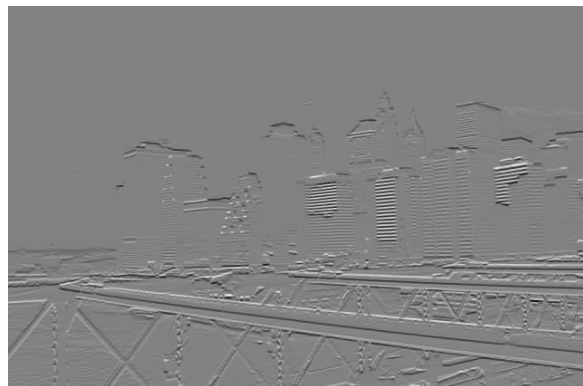
5.4.2 Sobel Edge Detector

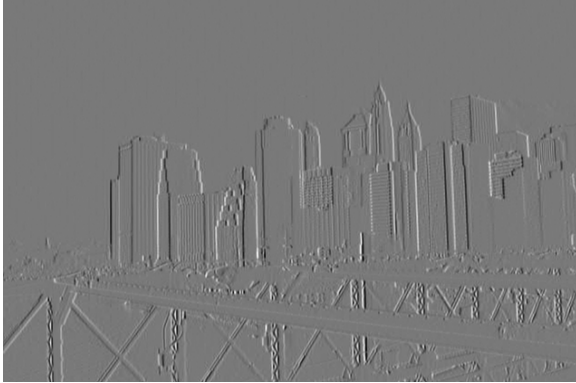
Description: The Sobel edge detector is the most common method for edge detection in an image. The Sobel method is more robust than the Robert's Cross and tends to find better edges. Like the Robert's Cross, the Sobel edge detector uses a series of two convolutions on the original image.

The Sobel kernels are shown below:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Because of the larger kernel size and the fact that edges are required to be at least 2 pixels wide to be detected, single pixel noise doesn't get detected. This results in a cleaner gradient edge. An example is shown below. The top-left image is the original, the top-right is the result from the first convolution, and the bottom-left is the result of the second convolution.





Again, like the Robert's Cross, these images can be combined to form a more complete gradient image. The bottom-right image shows the results from the RSS of the two edge images. The Sobel edge image is cleaner and has brighter edges than the Roberts Cross.

5.4.3 Prewitt / Compass Edge Detector

[TBD]

5.4.4 Quadratic Edge Detector

[TBD]

5.4.5 Canny Edge Detector

[TBD]

5.5 Boolean Mask Operators

Boolean mask operations are simple logical operations that are done on a pair of Boolean masks. These techniques are useful when working with morphological operators.

5.5.1 Boolean AND

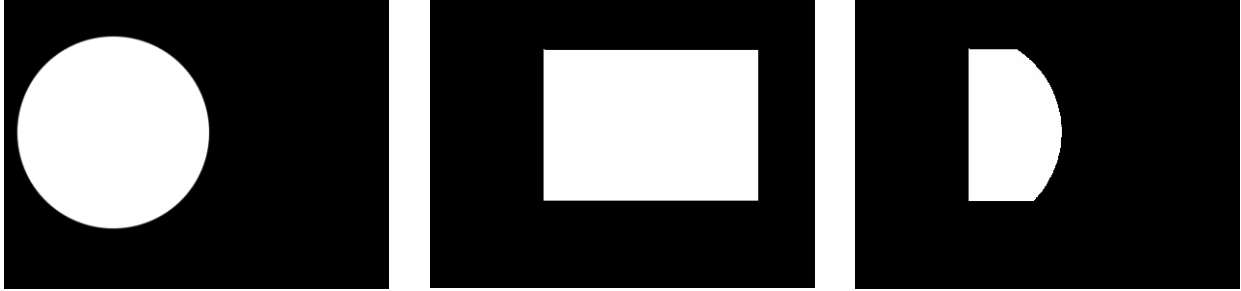
Description: Takes a pixel-by-pixel logical AND of two same sized masks.

The AND operation is a built in operation for a computer, so no additional logic is required. The equation for an AND is shown below along with its truth table. The operator \wedge represents the logical AND.

$$C = A \wedge B$$

A	B	C
T	T	T
T	F	F
F	T	F
F	F	F

The images below show an example of an AND operation. The left and middle images represent the input masks. The white regions are true and the black are false. The result of the logical AND is shown in the third image.



5.5.2 Boolean OR

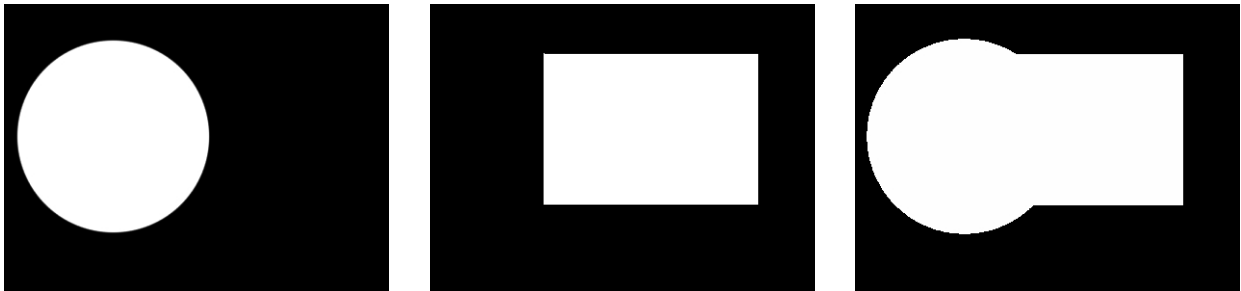
Description: Takes a pixel-by-pixel logical OR of two same sized masks.

The OR operation is a built in operation for a computer, so no additional logic is required. The equation for an OR is shown below along with its truth table. The operator \vee represents the logical OR.

$$C = A \vee B$$

A	B	C
T	T	T
T	F	T
F	T	T
F	F	F

The images below show an example of an OR operation. The left and middle images represent the input masks. The white regions are true and the black are false. The result of the logical OR is shown in the third image.



5.5.3 Boolean XOR

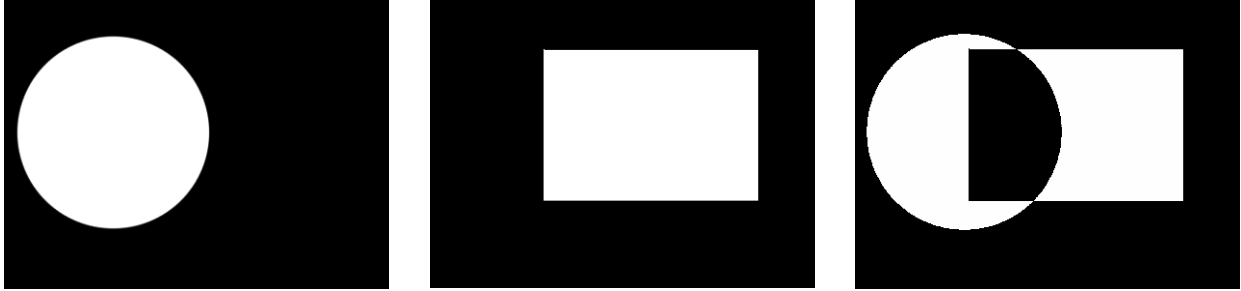
Description: Takes a pixel-by-pixel logical Exclusive OR of two same sized masks.

The XOR operation is built from a mixture of ANDs, ORs, and NOTs. The equation is shown below along with its truth table. The operators \wedge , \vee , and $!$ represent AND, OR, and NOT respectively.

$$C = (A \wedge !B) \vee (!A \wedge B)$$

A	B	C
T	T	F
T	F	T
F	T	T
F	F	F

The images below show an example of an XOR operation. The left and middle images represent the input masks. The white regions are true and the black are false. The result of the logical XOR is shown in the third image.



5.5.4 Boolean NOT

Description: Takes a pixel-by-pixel logical NOT of a mask.

The OR operation is a built in operation for a computer, so no additional logic is required. The equation is shown below along with its truth table. The operator ! represents a logical NOT.

$$B = !A$$

A	B
T	F
F	T

The images below show an example of a NOT operation. The left image represents the input mask. The white regions are true and the black are false. The result of the logical NOT is shown in the right image.



5.5.5 Boolean NAND

Description: Takes a pixel-by-pixel logical NAND of two same sized masks.

The NAND operation is the NOT of an AND operation. The equation is shown below along with its truth table. The operators \wedge and ! represent AND and NOT respectively.

$$C = !(A \wedge B)$$

A	B	C
T	T	F
T	F	T
F	T	T
F	F	T

The images below show an example of a NAND operation. The left and middle images represent the input masks. The white regions are true and the black are false. The result of the logical NAND is shown in the third image.



5.5.6 Boolean NOR

Description: Takes a pixel-by-pixel logical NOR of two same sized masks.

The NAND operation is the NOT of an AND operation. The equation is shown below along with its truth table. The operators \wedge and $!$ represent AND and NOT respectively.

$$C = !(A \vee B)$$

A	B	C
T	T	F
T	F	F
F	T	F
F	F	T

The images below show an example of a NOR operation. The left and middle images represent the input masks. The white regions are true and the black are false. The result of the logical NOR is shown in the third image.



5.5.7 Boolean XNOR

Description: Takes a pixel-by-pixel logical Exclusive NOR of two same sized masks.

The XNOR operation is built from a mixture of ANDs, ORs, and NOTs. The equation is shown below along with its truth table. The operators \wedge , \vee , and $!$ represent AND, OR, and NOT respectively.

$$C = (A \wedge B) \vee (!A \wedge !B)$$

A	B	C
T	T	T
T	F	F
F	T	F
F	F	T

The images below show an example of an XNOR operation. The left and middle images represent the input masks. The white regions are true and the black are false. The result of the XNOR is shown in the third image.



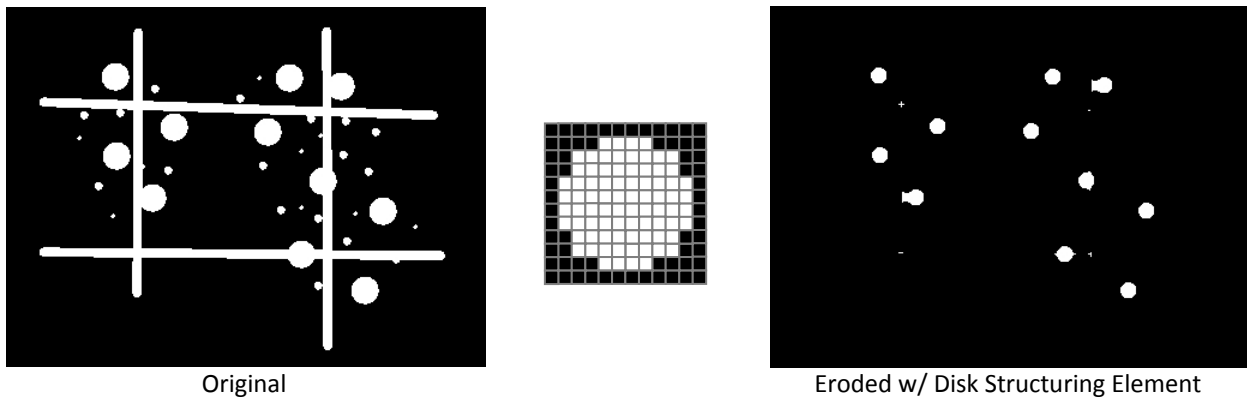
5.6 Morphological Operators

Morphological operators are special types of convolution kernels that can be used on any type of imagery, but are more typical on binary and grayscale images. Morphology is useful in feature extraction, image segmentation, and filtering artifacts from imagery.

5.6.1 Erosion

Description: Erosion is a process that removes a pixel from a region if its neighborhood doesn't match the shape of the kernel. The process is performed by passing the kernel over the image like a convolution. If the kernel and the underlying pixels at the current location match, the pixel returned to the new image is true, otherwise the pixel is false. Erosion is also a core operator that is used in more complex morphological operations.

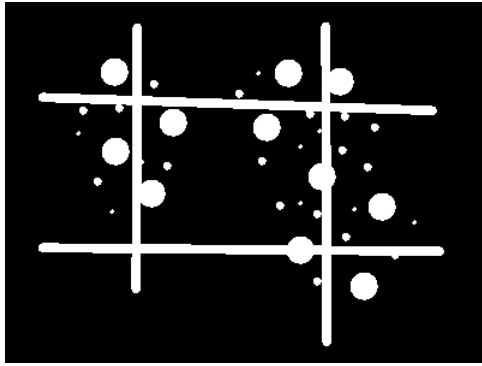
An example of erosion is shown in the binary image below. The left-hand image is the original, the middle image represents the erosion kernel, and the right image is the eroded version of the left.



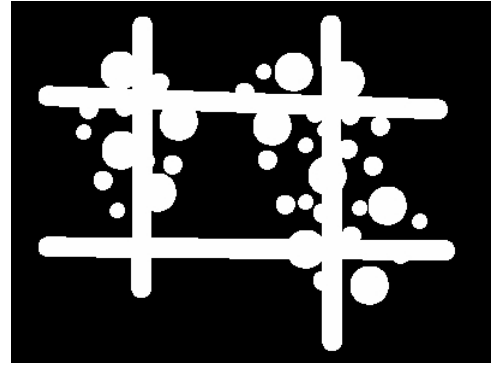
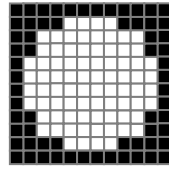
Different kernel shapes will yield different results. The erosion process keeps regions that are shaped like the kernel or that are large enough for the kernel to move around inside.

5.6.2 Dilation

[TBD]



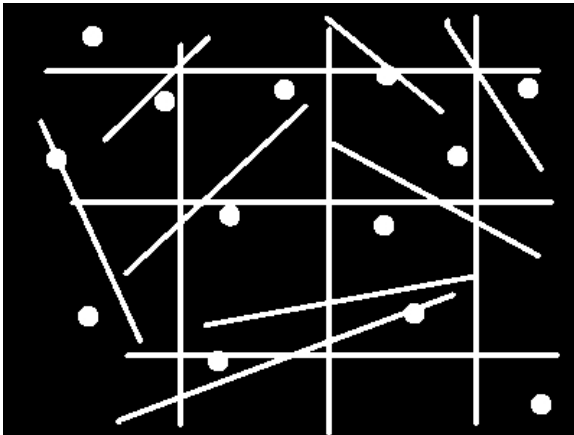
Original



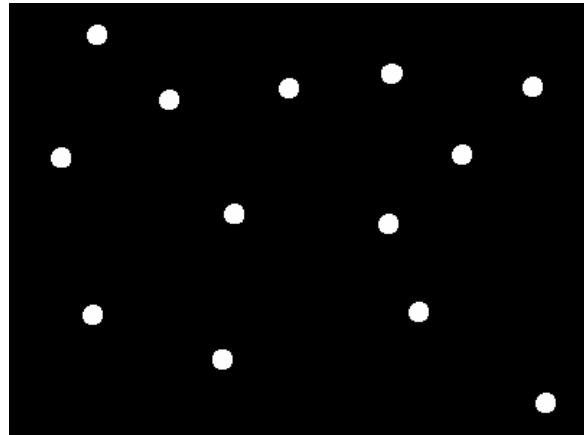
Dilated w/ Disk Structuring Element

5.6.3 Opening

[TBD]



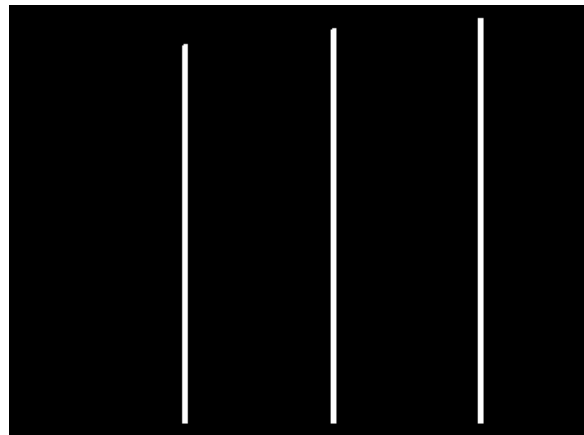
Original



Result from Opening w/
Disk Structuring Element

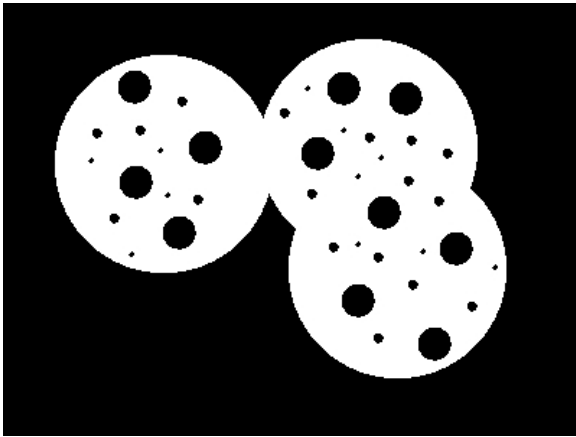


Result from Opening w/ Horizontal
Line Structuring Element

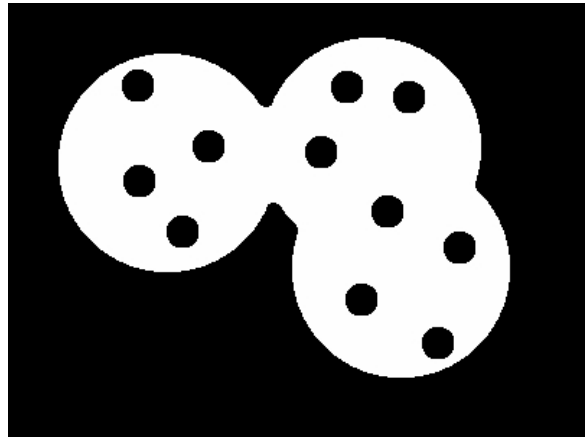


Result from Opening w/ Vertical
Line Structuring Element

5.6.4 Closing



Original



Result from Closing w/
Disk Structuring Element

[TBD]

5.6.5 Thinning

[TBD]



Original Image



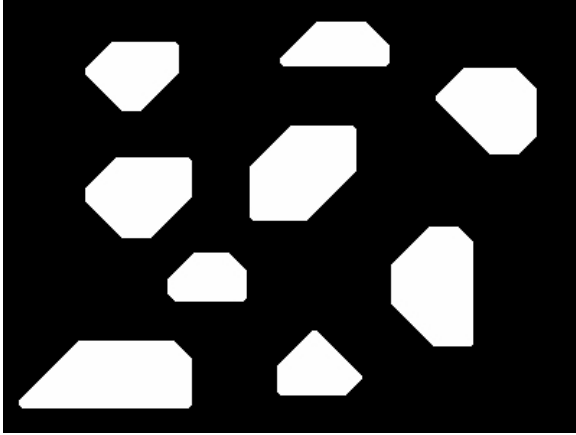
After 1 Pass



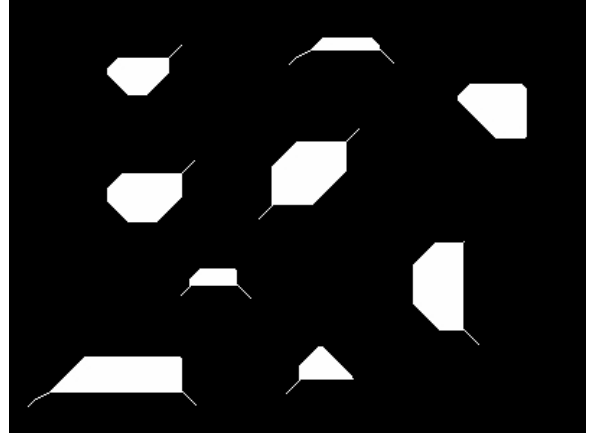
After 2 Passes



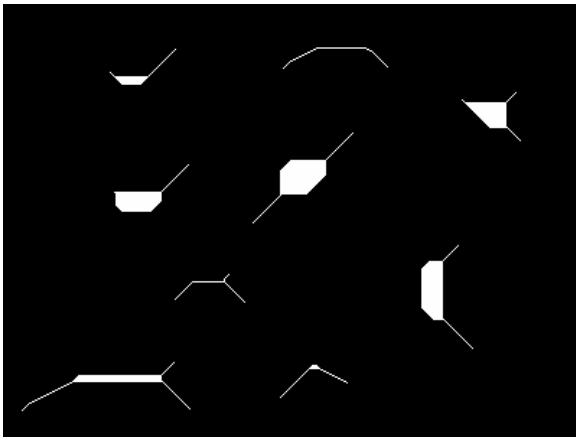
After 6 Passes (Complete)



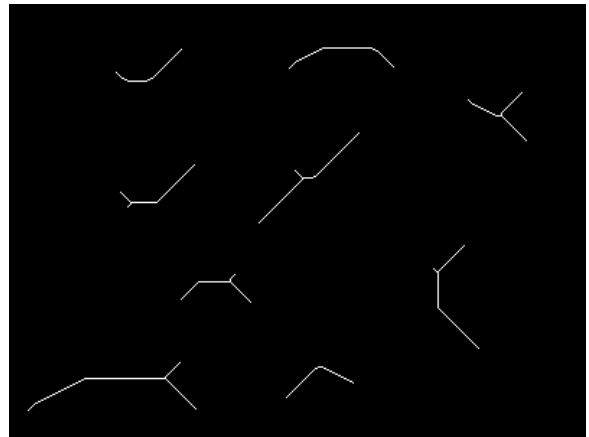
Original Image



After 10 Passes



After 20 Passes



After 33 Passes (Complete)

5.6.6 Thinning

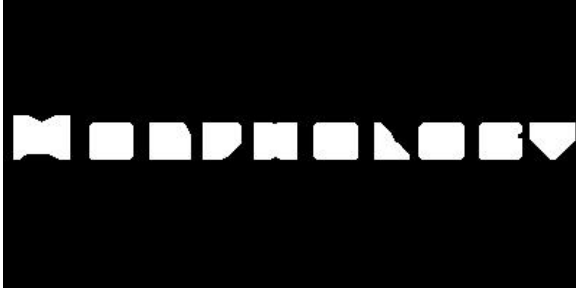
[TBD]



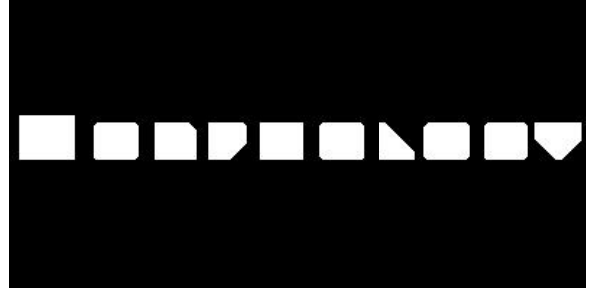
Original Image



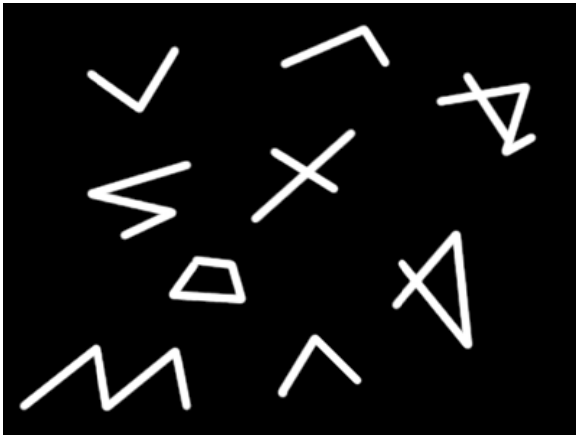
After 10 Passes



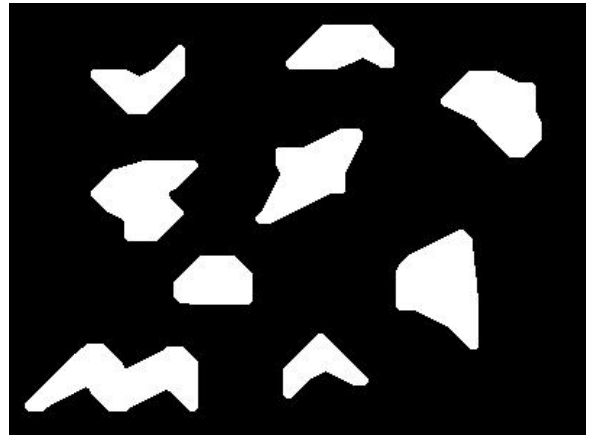
After 25 Passes



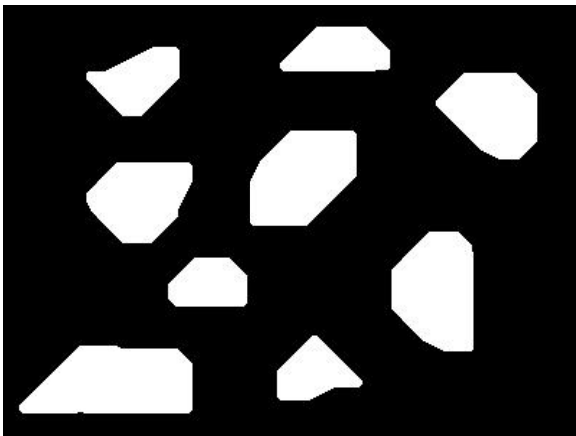
After 40 Passes (Complete)



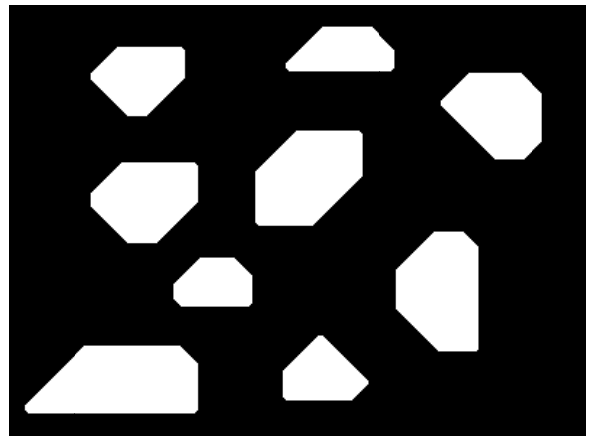
Original Image



After 30 Passes



After 60 Passes



After 102 Passes (Complete)

5.6.7 Hit or Miss Transform

[TBD]

5.6.8 Skeletonization

[TBD]

5.6.9 Pruning

[TBD]

5.6.10 Perimeter Detection

[TBD]

5.7 Image Enhancement Techniques

[TBD]

5.7.1 Linear Contrast Stretching

[TBD]

5.7.1.1 Linear Stretch

[TBD]

5.7.1.2 n-Sigma Stretch

[TBD]

5.7.2 Piecewise Contrast Stretch

[TBD]

5.7.3 Histogram Equalization

[TBD]

5.7.3.1 Uniform

[TBD]

5.7.3.2 Exponential

[TBD]

5.7.3.3 Gaussian

[TBD]

5.7.3.4 Hyperbolic

[TBD]

5.7.3.5 Log Hyperbolic

[TBD]

5.7.3.6 Rayleigh

[TBD]

5.7.4 Unsharp Masking

[TBD]

5.7.5 Binary Thresholding

[TBD]

5.7.6 Color Balancing

[TBD]

5.7.7 Gamma Correction

[TBD]

5.8 Texture Growth

[TBD]

5.9 Hough Transforms

[TBD]

5.10 Segmentation

[TBD]

5.11 Wavelets

[TBD]

5.12 References

- 1 Fisher, Robert, et. al., *The Hypermedia Image Processing Reference (HIPR2)*, "Morphology", "Gaussian Smoothing", http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm