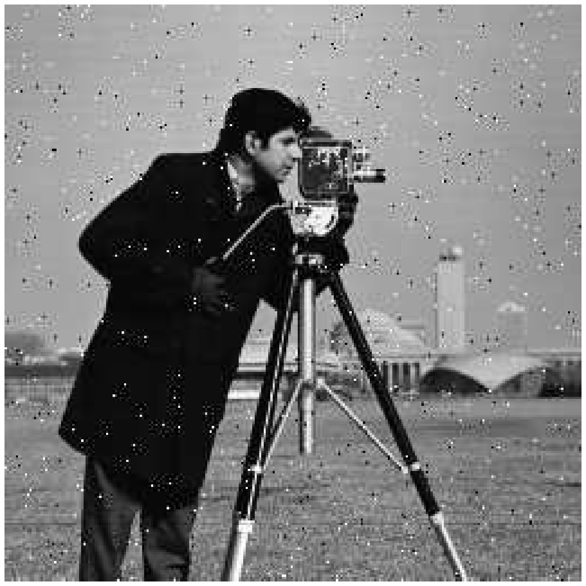
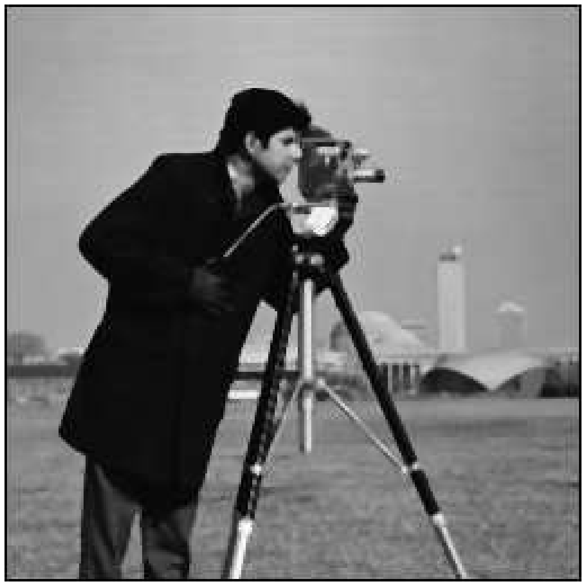
ECE 5273
HW 6 Solution

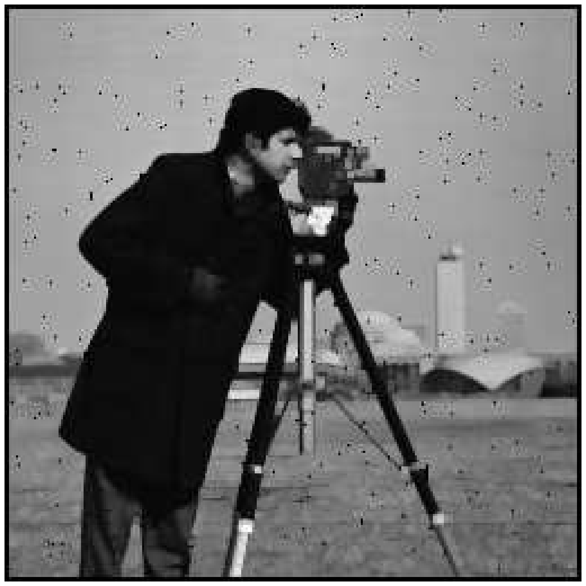
Spring 2025 Dr. Havlicek

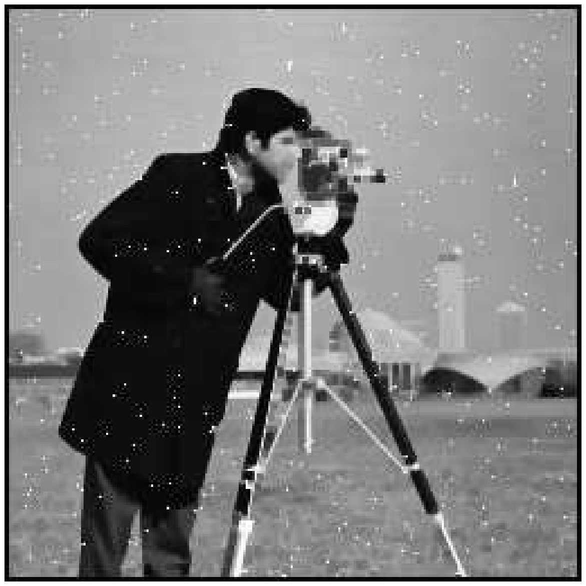
Note: This document contains solutions in both Matlab and traditional C.

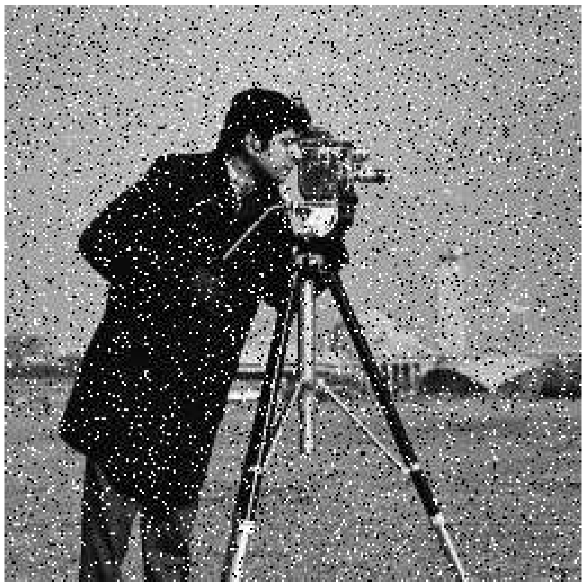
C Solution:

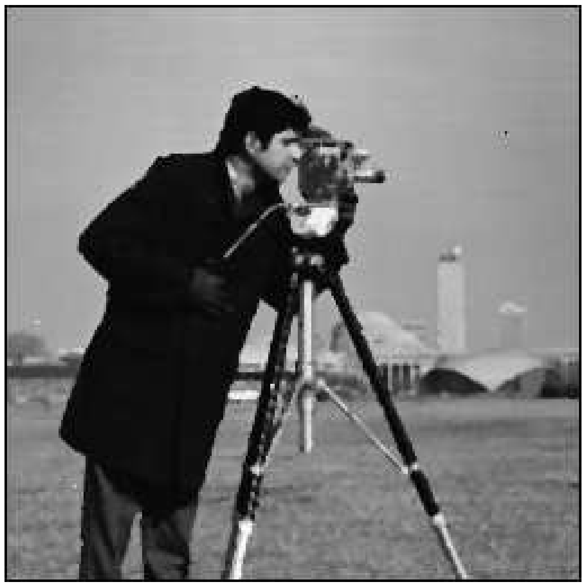

camera99.bin

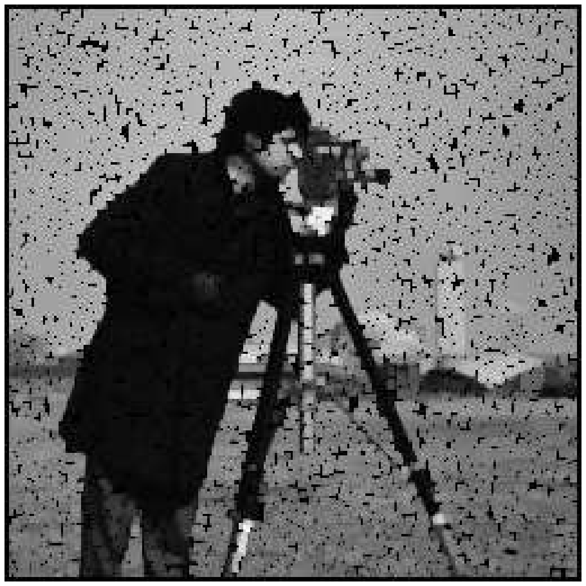

3×3 Median Filter

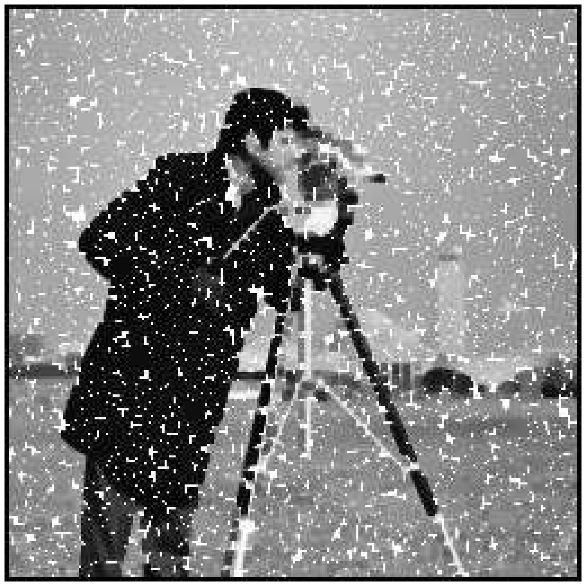

3×3 Morphological Opening


3×3 Morphological Closing


camera9.bin


3×3 Median Filter


3×3 Morphological Opening


3×3 Morphological Closing

Discussion

The results of applying the 3×3 median filter, morphological opening, and morphological closing to *camera99.bin* are shown on page 1. The image *camera99.bin* was obtained from *camera.bin* by corrupting the original *cameraman* image with salt-and-pepper noise, where the probability of a pixel being set to 255 was 0.005, the probability of a pixel being set to 0 was 0.005, and the probability of a pixel retaining its original gray level was 0.99.

The median filter removes both the positive-going and negative-going spikes. For *camera99.bin*, the 3×3 median filter is very effective for removing the salt-and-pepper noise. Note that some "blotching" artifacts are visible in the median filtered result — e.g., some detail is lost in the camera, the camera support gimbal, and the subject's facial features. However, overall this result is an excellent example of image restoration in the presence of a heavily-tailed additive noise.

The result of applying the 3×3 morphological opening to *camera99.bin* is similar to the median filtered result, except that only the *positive-going* spikes are removed, while some linking occurs between the negative-going spikes. Likewise, the result of applying the 3×3 morphological closing is also similar to the median filtered result, except that this time only the *negative-going* spikes are removed, while some linking occurs between the positive-going spikes.

Results for the corrupted image *camera9.bin* are shown on page 2. This image was again obtained by adding salt-and-pepper noise to *camera.bin*, except that this time the probability of a pixel being set to 255 or to 0 was ten times greater than in *camera99.bin*. Nevertheless, the 3×3 median filter still succeeds in removing nearly all of the salt-and-pepper noise. As before, performance of the OPEN and CLOSE operations is similar to the median filter, where the OPEN filter removes only the positive-going spikes while linking some of the negative-going spikes, whereas the CLOSE filter removes only the negative-going spikes while linking some of the positive-going spikes.

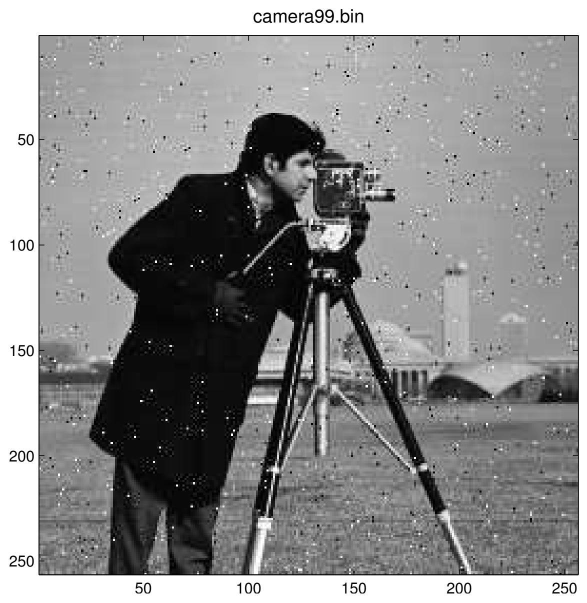
C program listing:

```
/*
 * hw06.c:
 *
 *  Apply 3x3 MF, OPEN, and CLOSE to a gray scale byte image.
 *  Handle edge effects by making the output zero.
 *
 *  The input image is assumed square.
 *
 *
 * 04/25/2006 jph
 *
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>

#define BYTE unsigned char

/*
 * Function prototypes
 */
 void  disk2byte();
 void  byte2disk();


/*----------------------------------------------------------------------*/
/*   MAIN                                                               */
/*----------------------------------------------------------------------*/

main(argc,argv)

  int     argc;
  char   *argv[];
{

  int      size;            /* num rows/cols in image */
  char    *infile;          /* input filename */
  char    *MFoutfile;       /* output filename for median filtered image */
  char    *Ooutfile;        /* output filename for OPENed image */
  char    *Coutfile;        /* output filename for CLOSEed image */
  BYTE    *x;               /* input image */
  BYTE    *yMF;             /* median filtered output image */
  BYTE    *yO;              /* OPENed output image */
  BYTE    *yC;              /* CLOSEed output image */
  BYTE    *yD;              /* DILATEed image */
  BYTE    *yE;              /* ERODEed image */
  BYTE    *W;               /* the windowed set of pixels */
  int      wsize;           /* window size */
  int      wsizeo2;         /* window size divided by 2 */
  int      Max;             /* max element of window set */
  int      middle;          /* index in sort(W) of the median */
  int      min;             /* min element of window set */
  BYTE     temp;            /* temporary place holder for sorting */
  int      i,j;             /* loop counters */
  int      row,col;         /* loop counters for image rows and cols */

 /*
  * Parse args and check for proper invocation
  */
  if (argc != 7) {
    printf("\n%s: Apply w x w MF, OPEN, and CLOSE to a BYTE image.",
           argv[0]);
    printf("\nUsage: %s size w infile MFoutfile Ooutfile Coutfile\n",argv[0]);
    exit(0);
  }
  size = atoi(argv[1]);
  wsize = atoi(argv[2]);
  infile = argv[3];
  MFoutfile = argv[4];
  Ooutfile = argv[5];
  Coutfile = argv[6];

  if (!(wsize%2)) {
    printf("\n%s: w (window size) must be ODD!\n",argv[0]);
    exit(0);
  }
  wsizeo2 = wsize >> 1;
  middle = (wsize*wsize) >> 1;

 /*
  * Allocate image arrays
  */
  if ((x = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((yMF = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((yO = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((yC = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((yD = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((yE = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((W = (BYTE *)malloc(wsize*wsize*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }

 /*
  * Initialize images
  */
  for (i=0; i < size*size; i++) {
    yMF[i] = yO[i] = yC[i] = yD[i] = yE[i] = 0;
  }

 /*
  * Read the input image from disk
  */
  disk2byte(x,size,size,infile);

 /*
  * Apply Median Filter, Erode, and Dilate
  */
  for (row=wsizeo2; row < size-wsizeo2; row++) {
    for (col=wsizeo2; col < size-wsizeo2; col++) {
      for (j=-wsizeo2; j <= wsizeo2; j++) {
        for (i=-wsizeo2; i <= wsizeo2; i++) {
          W[(j+wsizeo2)*wsize + i + wsizeo2] = x[(row+j)*size + col+i];
        }    /* for i */
      }      /* for j */

     /*
```

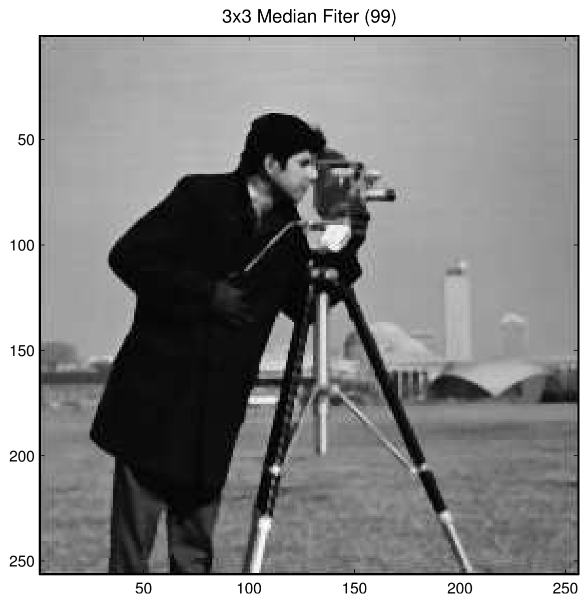
```
       * Sort the window set (full sort needed for OPEN and CLOSE)
       */
       for (i=0; i < wsize*wsize; i++) {
         for (j=i+1; j < wsize*wsize; j++) {
           if (W[j] < W[i]) {
             temp = W[j];
             W[j] = W[i];
             W[i] = temp;
           }
         }    /* for j */
       }      /* for i */

      /*
       * Median Filter
       */
       yMF[row*size + col] = W[middle];

      /*
       * Dilate
       */
       yD[row*size + col] = W[wsize*wsize - 1];

      /*
       * Erode
       */
       yE[row*size + col] = W[0];
     }    /* for col */
   }      /* for row */

  /*
   * Finish the OPEN by applying DILATE to yE
   */
   for (row=wsizeo2+1; row < size-wsizeo2-1; row++) {
     for (col=wsizeo2+1; col < size-wsizeo2-1; col++) {
       for (j=-wsizeo2; j <= wsizeo2; j++) {
         for (i=-wsizeo2; i <= wsizeo2; i++) {
           W[(j+wsizeo2)*wsize + i +wsizeo2] = yE[(row+j)*size + col+i];
         }    /* for i */
       }      /* for j */

      /*
       * Find MAX of the window set
       */
       Max = W[0];
       for (i=1; i < wsize*wsize; i++) {
         if (Max < W[i]) {
           Max = W[i];
         }
       }    /* for i */

      /*
       * DILATE
       */
       yO[row*size + col] = Max;
     }    /* for col */
   }      /* for row */



  /*
   * Finish the CLOSE by applying ERODE to yD
   */
   for (row=wsizeo2+1; row < size-wsizeo2-1; row++) {
     for (col=wsizeo2+1; col < size-wsizeo2-1; col++) {
       for (j=-wsizeo2; j <= wsizeo2; j++) {
         for (i=-wsizeo2; i <= wsizeo2; i++) {
           W[(j+wsizeo2)*wsize + i +wsizeo2] = yD[(row+j)*size + col+i];
         }    /* for i */
       }      /* for j */
      /*
       * Find min of the window set
       */
       min = W[0];
       for (i=1; i < wsize*wsize; i++) {
         if (min > W[i]) {
           min = W[i];
         }
       }    /* for i */

      /*
       * ERODE
       */
       yC[row*size + col] = min;
     }    /* for col */
   }      /* for row */

  /*
   * Write the output images to disk
   */
   byte2disk(yMF,size,size,MFoutfile);
   byte2disk(yO,size,size,Ooutfile);
   byte2disk(yC,size,size,Coutfile);

   return;

 } /*----------------   MAIN   ------------------------------------------*/

 /*----------------------------------------------------------------------
  * disk2byte.c
  *
  *   function reads an unsigned char (byte) image from disk
  *
  *
  *  jph 15 June 1992
  *
  -----------------------------------------------------------------------*/

 void disk2byte(x,row_dim,col_dim,fn)

   BYTE    *x;             /* image to be read */
   int     row_dim;        /* row dimension of x */
   int     col_dim;        /* col dimension of x */
   char   *fn;             /* filename */
 {

   int fd;               /* file descriptor */
   int n_bytes;          /* number of bytes to read */

  /*
   * detect zero dimension input
   */
   if ((row_dim==0) || (col_dim==0)) return;

  /*
   * create and open the file
   */
   if ((fd = open(fn, O_RDONLY))==-1) {
     printf("\ndisk2byte.c : could not open %s !",fn);
     return;
   }

  /*
   * read image data from the file
   */
   n_bytes = row_dim * col_dim * sizeof(unsigned char);
   if (read(fd,x,n_bytes) != n_bytes) {
     printf("\ndisk2byte.c : complete read of %s did not succeed.",fn);
   }
  /*
   * close file and return
   */
   if (close(fd) == -1) printf("\ndisk2byte.c : error closing %s.",fn);
   return;
 } /*---------------------  disk2byte  ---------------------------------*/

 /*-------------------------------------------------------------------
  * byte2disk.c
  *
  *   function writes an unsigned char (byte) image to disk
  *
  *
  *  jph 15 June 1992
  *
  ----------------------------------------------------------------------*/

 void byte2disk(x,row_dim,col_dim,fn)

   BYTE  *x;              /* image to be written */
   int   row_dim;         /* row dimension of x */
   int   col_dim;         /* col dimension of x */
   char *fn;              /* filename */
 {

   int fd;               /* file descriptor */
   int n_bytes;          /* number of bytes to read */

  /*
   * detect zero dimension input
   */
   if ((row_dim==0) || (col_dim==0)) return;

  /*
   * create and open the file
   */
   if ((fd = open(fn, O_WRONLY | O_CREAT | O_TRUNC, 0644))==-1) {
     printf("\nbyte2disk.c : could not open %s !",fn);
     return;
   }

  /*
   * write image data to the file
   */
   n_bytes = row_dim * col_dim * sizeof(unsigned char);
   if (write(fd,x,n_bytes) != n_bytes) {
     printf("\nbyte2disk.c : complete write of %s did not succeed.",fn);
   }

  /*
   * close file and return
   */
   if (close(fd) == -1) printf("\nbyte2disk.c : error closing %s.",fn);
   return;
 } /*------------------------  byte2disk  --------------------------------*/
```

Matlab Solution:

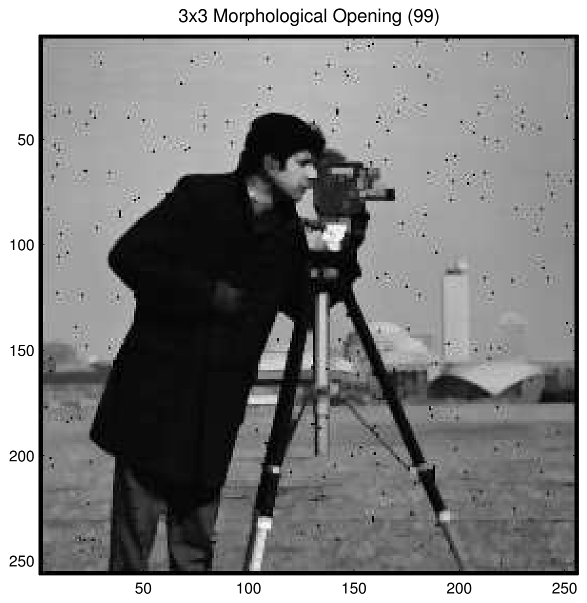
Note: Explanations are not given in the Matlab solution. The explanations are the same as the ones already given above in the C solution.



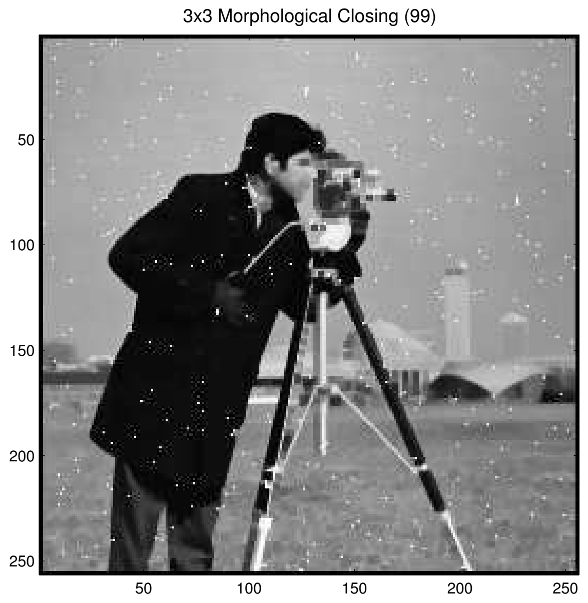
camera99.bin



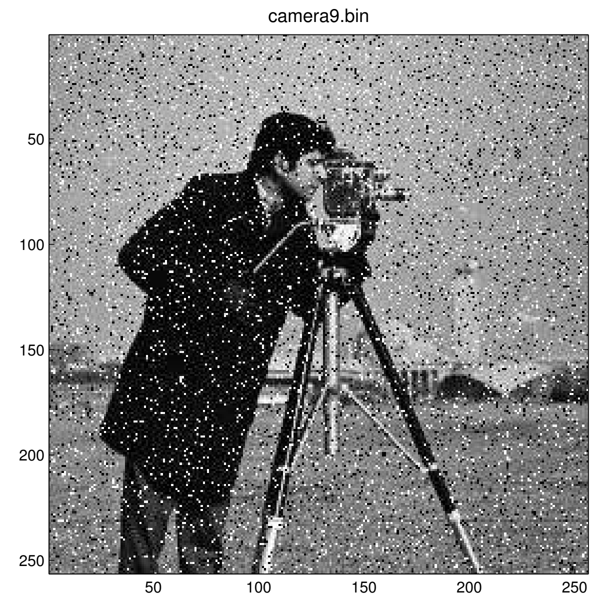
3×3 Median Filter



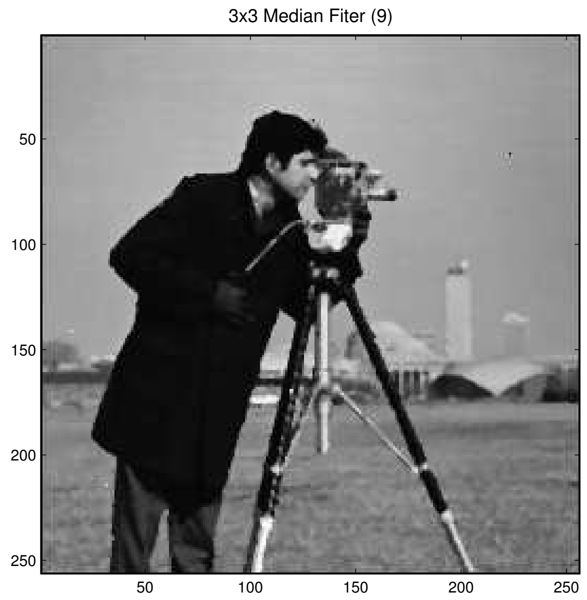
3×3 Morphological Opening



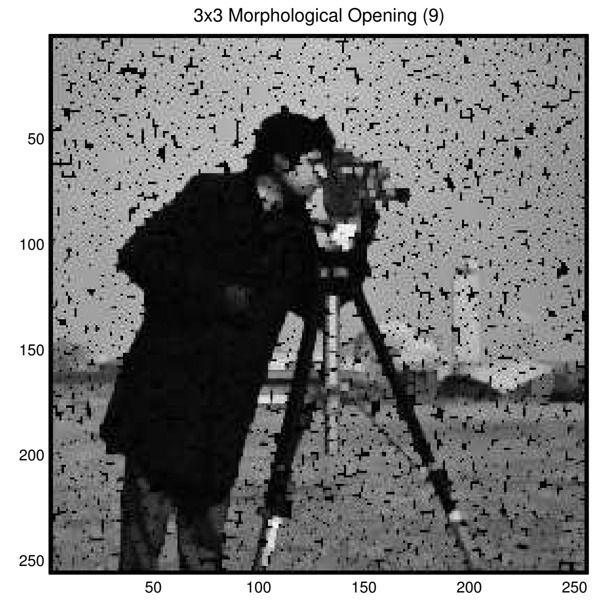
3×3 Morphological Closing



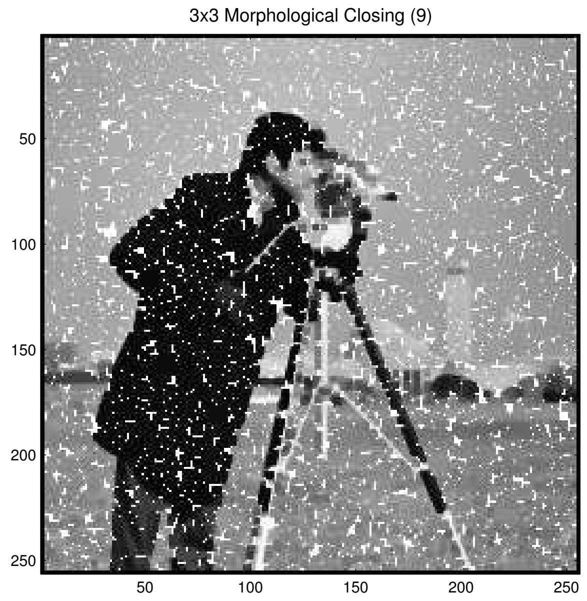
camera9.bin



3×3 Median Filter



3×3 Morphological Opening



3×3 Morphological Closing

Matlab m-file listing:

```
%
% hw06.m
%
%   Apply 3x3 MF, OPEN, and CLOSE to a gray scale byte image.
%   Handle edge effects by making the output zero.
%
%   The input image is assumed square.
%
%   NOTE: in the code below, W will hold the window set of pixel values.
%         - median(W(:)) will correctly find the median of the window set.
%         - median(W) is NOT correct.  This returns a vector of the column
%              medians of W.
%         - median(median(W)) is also NOT correct.  It implements something
%           called a separable median filter, which is NOT the same thing
%           as the median filter.
%
%
% 04/25/06 jph
%


%-------------------------------------------------------------------
% Initialize
%
clear;
size = 256;                        % number of rows/cols in the image
wsize = 3;                         % for 3x3 filter windows
wsizeo2 = fix(wsize/2);            % window half-size ( =1 for 3x3)
FigNum = 1;                        % initialize the figure number

W = zeros(wsize);                  % filter window

yMF = zeros(size);                 % median filtered image
yE = zeros(size);                  % eroded image
yD = zeros(size);                  % dilated image
yO = zeros(size);                  % opened image
yC = zeros(size);                  % closed image


%
% camera99.bin
%
fprintf(1,'Doing camera99.bin...\n');

%
% Read and display the image
%
x = ReadBin('camera99.bin',size);
FigNum = ShowByteImage(x,FigNum);
title('camera99.bin','FontSize',12);
print -deps camera99.eps;

%
% Apply median, erode, and dilate
%
for row=wsizeo2+1:size-wsizeo2
  for col=wsizeo2+1:size-wsizeo2
    W = x(row-wsizeo2:row+wsizeo2,col-wsizeo2:col+wsizeo2);
    yMF(row,col) = median(W(:));
    yE(row,col) = min(W(:));
    yD(row,col) = max(W(:));
  end
end

%
% Apply dilate to yE and erode to yD to finish the open and close
%
for row=wsizeo2+2:size-wsizeo2-1
```

```
  for col=wsizeo2+2:size-wsizeo2-1
    W = yE(row-wsizeo2:row+wsizeo2,col-wsizeo2:col+wsizeo2);
    yO(row,col) = max(W(:));
    W = yD(row-wsizeo2:row+wsizeo2,col-wsizeo2:col+wsizeo2);
    yC(row,col) = min(W(:));
  end
end

%
% Display the output images
%
FigNum = ShowByteImage(yMF,FigNum);
title('3x3 Median Fiter (99)','FontSize',12);
print -deps camera99MF.eps;

FigNum = ShowByteImage(yO,FigNum);
title('3x3 Morphological Opening (99)','FontSize',12);
print -deps camera99O.eps;

FigNum = ShowByteImage(yC,FigNum);
title('3x3 Morphological Closing (99)','FontSize',12);
print -deps camera99C.eps;



%--------------------------------------------------------------------
% Do it all over again for camera9.bin
%
fprintf(1,'\nHit any key to continue...\n');
pause;
size = 256;                         % number of rows/cols in the image
wsize = 3;                          % for 3x3 filter windows
wsizeo2 = fix(wsize/2);             % window half-size ( =1 for 3x3)

W = zeros(wsize);                   % filter window

yMF = zeros(size);                  % median filtered image
yE = zeros(size);                   % eroded image
yD = zeros(size);                   % dilated image
yO = zeros(size);                   % opened image
yC = zeros(size);                   % closed image

fprintf(1,'Doing camera9.bin...\n');

%
% Read and display the image
%
x = ReadBin('camera9.bin',size);
FigNum = ShowByteImage(x,FigNum);
title('camera9.bin','FontSize',12);
print -deps camera9.eps;

%
% Apply median, erode, and dilate
%
for row=wsizeo2+1:size-wsizeo2
  for col=wsizeo2+1:size-wsizeo2
    W = x(row-wsizeo2:row+wsizeo2,col-wsizeo2:col+wsizeo2);
    yMF(row,col) = median(W(:));
    yE(row,col) = min(W(:));
    yD(row,col) = max(W(:));
  end
end

%
% Apply dilate to yE and erode to yD to finish the open and close
%
for row=wsizeo2+2:size-wsizeo2-1
  for col=wsizeo2+2:size-wsizeo2-1
    W = yE(row-wsizeo2:row+wsizeo2,col-wsizeo2:col+wsizeo2);
    yO(row,col) = max(W(:));
    W = yD(row-wsizeo2:row+wsizeo2,col-wsizeo2:col+wsizeo2);
    yC(row,col) = min(W(:));
  end
end

%
% Display the output images
%
FigNum = ShowByteImage(yMF,FigNum);
title('3x3 Median Fiter (9)','FontSize',12);
print -deps camera9MF.eps;

FigNum = ShowByteImage(yO,FigNum);
title('3x3 Morphological Opening (9)','FontSize',12);
print -deps camera9O.eps;

FigNum = ShowByteImage(yC,FigNum);
title('3x3 Morphological Closing (9)','FontSize',12);
print -deps camera9C.eps;

fprintf(1,'\n');
return;


%
% ReadBin.m
%
%   Read a square raw BYTE image (one byte per pixel, no header) from disk
%   into a matlab array.
%
%   Usage:
%   >> x = ReadBin(fn,size);
%
%   Input parameters:
%     fn         input filename
%     size       number of rows/cols in the image
%
%   Output parameters:
%     x          double output array, holds the image
%
% 4/3/03 jph
%

function [x] = ReadBin(fn,size)

%
% Open the file
%
fid = fopen(fn,'r');
if (fid == -1)
  error(['Could not open ',fn]);
end;

%
% Read and close the file
%
[x,Nread] = fread(fid,[size,size],'uchar');
if (Nread ~= size*size)
  error(['Complete read of ',fn,' did not succeed.']);
end;
fclose(fid);

%
% Transpose data for matlab's 'row major' convention and return
%
x = x';


%
```

```
% ShowByteImage.m
%
%   Display a grayscale byte-per-pixel image.
%
%   NOTE: does NOT automatically do a full-scale stretch.
%
%   Usage:
%   >> FigNum = ShowByteImage(x,FigNum);
%   >> ShowByteImage(x,FigNum);
%
%   Input parameters:
%     x          double array holding the byte image to be displayed
%     FigNum     number of the figure in which to display the image
%
%   Output parameters:
%     FigNum     the value of FigNum is incremented prior to return.
%                this makes an optional but convenient way to generate
%                consecutively numbered figures with iterated calls to
%                this function.
%
% 4/3/03 jph
%

function [NewFigNum] = ShowByteImage(x,FigNum)

figure(FigNum);
colormap(gray(256));
image(x);
axis('image');

NewFigNum = FigNum + 1;
```