

Percobaan 1

```
%--membaca file citra dan template--%
>> img=imread('text.png');
>> img1=imread('D:\Ayu Documents\Gambar\text1.png'); %--karena template
tidak berada pada direktori Matlab maka lokasi template pada sistem harus
dispesifikasi--%
%--menampilkan citra dan template--%
>> imshow(img)
>> figure,imshow(img1)
%--pengolahan citra--%
>>img_f3= tm(img,img1);
%menampilkan citra hasil pengolahan sebelum dithreshold--%
>> figure,imshow(img_f3,[])
%--mencari nilai maksimum--%
>> max(img_f3(:))

ans =

    68
%--tentukan nilai threshold--%
>> thresh=60;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)
```

Percobaan 2

```
%--membaca file citra dan template--%
>> img=imread('D:\Ayu Documents\Gambar\buah.png');
>> img1=imread('D:\Ayu Documents\Gambar\buah2.png'); %--karena template
tidak berada pada direktori Matlab maka lokasi template pada sistem harus
dispesifikasi--%

%--menampilkan citra dan template--%
>> imshow(img)
>> figure,imshow(img1)
%--pengolahan citra--%
>>img_f3= tm(img,img1);
%menampilkan citra hasil pengolahan sebelum dithreshold--%
>> figure,imshow(img_f3,[])
%--mencari nilai maksimum--%
>> max(img_f3(:))

ans =
```

20328842

```
%--tentukan nilai threshold--%
>> thresh=20028842;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)
```

Percobaan 3

```
%--membaca file citra dan template--%
>> img=imread('D:\Ayu Documents\Gambar\Gambar Baru\ch_tree.png');
>> img1=imread('D:\Ayu Documents\Gambar\Gambar Baru\ch_tree1.png'); );
%--karena template tidak berada pada direktori Matlab maka lokasi template pada
sistem harus dispesifikasi--%
```

```
%--menampilkan citra dan template--%
>> imshow(img)
>> figure,imshow(img1)
%--pengolahan citra--%
>>img_f3= tm(img,img1);
%menampilkan citra hasil pengolahan sebelum dithreshold--%
>> figure,imshow(img_f3,[])
%--mencari nilai maksimum--%
>> max(img_f3(:))
```

ans =

37434150

```
%--tentukan nilai threshold--%
>>thresh=37034150;%--tampilkan citra hasil pengolahan yang sudah
dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)
```

Percobaan 4

```
>> img=imread('D:\Ayu Documents\Gambar\Gambar Baru\ch_tree.png');
>> img1=imread('D:\Ayu Documents\Gambar\Gambar Baru\ch_tree2.png'); %--
-karena template tidak berada pada direktori Matlab maka lokasi template pada
sistem harus dispesifikasi--%
```

```
%--menampilkan citra dan template--%
>> imshow(img)
>> figure,imshow(img1)
%--pengolahan citra--%
>>img_f3= tm(img,img1);
```

```

%menampilkan citra hasil pengolahan sebelum dithreshold--%
>> figure,imshow(img_f3,[])
%--mencari nilai maksimum--%
>> max(img_f3(:))

ans =

    36717204

%--tentukan nilai threshold--%
>> thresh=36017204;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)

```

Percobaan 5

```

>> img=imread('D:\Ayu Documents\Gambar\Gambar Baru\lamp.png');
>> img1=imread('D:\Ayu Documents\Gambar\Gambar Baru\lamp1.png'); ) %--
-karena template tidak berada pada direktori Matlab maka lokasi template pada
sistem harus dispesifikasi--%

```

```

%--menampilkan citra dan template--%
>> imshow(img)
>> figure,imshow(img1)
%--pengolahan citra--%
>>img_f3= tm(img,img1);
%menampilkan citra hasil pengolahan sebelum dithreshold--%
>> figure,imshow(img_f3,[])
%--mencari nilai maksimum--%
>> max(img_f3(:))

```

ans =

76895989

```

%--tentukan nilai threshold--%
>> thresh=76095989;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)
%--tentukan nilai threshold--%
>> thresh=76795989;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)
%--tentukan nilai threshold--%
>> thresh=75895989;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%

```

```

>> figure,imshow(img_f3>thresh)
%--tentukan nilai threshold--%
>> thresh=76895089;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)
%--tentukan nilai threshold--%
>> thresh=74695989;
%--tampilkan citra hasil pengolahan yang sudah dithreshold(lokalasi diketahui)--%
>> figure,imshow(img_f3>thresh)

```

```

function x = tm(A,B)
% Class support for input F:
% float: double, single
%example
%A=imread('text.png');
%B=imread('text1.png');
%C=tm(A,B);
% Built-in function.
threeD = (ndims(A)==3);
if threeD
    A=rgb2gray(A);
    B=rgb2gray(B);
    m=size(A,1);
    n=size(A,2);
%--transformasi Fourier--%
    img_f=fft2(double(A));
    img_f1=fft2(double(B),m,n); %--size dari img1 dibuat sama dengan size img,
ubah tipe kelasnya menjadi double--%
%--perkalian matrik--%
    img_f2=img_f.*img_f1;
%--inverse Transformasi Fourier--%
    x=real(iff2(img_f2));
else
    m=size(A,1);
    n=size(A,2);
%--transformasi Fourier--%
    img_f=fft2(double(A));
    img_f1=fft2(double(B),m,n); %--size dari img1 dibuat sama dengan size img,
ubah tipe kelasnya menjadi double--%
%--perkalian matrik--%
    img_f2=img_f.*img_f1;
%--inverse Transformasi Fourier--%
    x=real(iff2(img_f2));
end

```

```

function [X, map, alpha] = imread(varargin)
%IMREAD Read image from graphics file.
% A = IMREAD(FILENAME,FMT) reads a grayscale or color image from the
% file specified by the string FILENAME, where the string FMT specifies
% the format of the file. See the reference page, or the output of the
% function IMFORMATS, for a list of supported formats. If the file is
% not in the current directory or in a directory in the MATLAB path,
% specify the full pathname of the location on your system. If IMREAD
% cannot find a file named FILENAME, it looks for a file named
% FILENAME.FMT.
%
% IMREAD returns the image data in the array A. If the file contains a
% grayscale image, A is a two-dimensional (M-by-N) array. If the file
% contains a color image, A is a three-dimensional (M-by-N-by-3)
% array. The class of the returned array depends on the data type used
% by the file format.
%
% For most file formats, the color image data returned uses the RGB
% color space. For TIFF files, however, IMREAD can return color data
% that uses the RGB, CIELAB, ICCLAB, or CMYK color spaces. If the
% color image uses the CMYK color space, A is an M-by-N-by-4 array.
% See the reference page for more information about reading TIFF files
% that use these color spaces.
%
% [X,MAP] = IMREAD(FILENAME,FMT) reads the indexed image in
FILENAME
% into X and its associated colormap into MAP. Colormap values in the
% image file are automatically rescaled into the range [0,1].
%
% [...] = IMREAD(FILENAME) attempts to infer the format of the file
% from its content.
%
% [...] = IMREAD(URL,...) reads the image from an Internet URL. The
% URL must include the protocol type (e.g., "http://").

% See also IMFINFO, IMWRITE, IMFORMATS, FREAD, IMAGE, DOUBLE,
UINT8.

% Steven L. Eddins, June 1996
% Copyright 1984-2004 The MathWorks, Inc.
% $Revision: 1.1.6.3 $ $Date: 2004/04/01 16:12:26 $

[filename, format, extraArgs, msg] = parse_inputs(varargin{:});
if (~isempty(msg))
    error('MATLAB:imread:inputParsing', '%s', msg);
end

```

```

% Download remote file.
if (strfind(filename, '://'))

    url = true;

    if (~usejava('mwt'))
        error('MATLAB:imread:noJava', 'Reading from a URL requires a Java
Virtual Machine.')
    end

    try
        filename = urlwrite(filename, tempname);
    catch
        error('MATLAB:imread:readURL', 'Can"t read URL "%s".', filename);
    end

else

    url = false;

end

if (isempty(format))
    % The format was not specified explicitly.

    % Verify that the file exists.
    fid = fopen(filename, 'r');
    if (fid == -1)

        if ~isempty(dir(filename))
            error('MATLAB:imread:fileOpen', ['Can"t open file "%s" for
reading;\nyou' ...
            ' may not have read permission.'], ...
            filename);
        else
            error('MATLAB:imread:fileOpen', 'File "%s" does not exist.', filename);
        end

    else
        % File exists. Get full filename.
        filename = fopen(fid);
        fclose(fid);
    end
end

```

```

% Try to determine the file type.
format = imftype(filename);

if (isempty(format))
    error('MATLAB:imread:fileFormat', 'Unable to determine the file format.');
```

end

```

else
    % The format was specified explicitly.

    % Verify that the file exists.
    fid = fopen(filename, 'r');
    if (fid == -1)
        % Couldn't open using the given filename; search for a
        % file with an appropriate extension.
        fmt_s = imformats(format);

        if (isempty(fmt_s))
            error('MATLAB:imread:fileFormat', ['Couldn't find format %s in the
format registry.' ...
            ' See "help imformats".'], format);
        end

        for p = 1:length(fmt_s.ext)
            fid = fopen([filename '.' fmt_s.ext{p}]);

            if (fid ~= -1)
                % The file was found. Don't continue searching.
                break
            end
        end
    end

    if (fid == -1)
        if ~isempty(dir(filename))
            error('MATLAB:imread:fileOpen', ['Can't open file "%s" for
reading;\nyou' ...
            ' may not have read permission.'], ...
            filename);
        else
            error('MATLAB:imread:fileOpen', 'File "%s" does not exist.', filename);
        end
    end
    filename = fopen(fid);
    fclose(fid);
end

```

```

end

% Get format details.
fmt_s = imformats(format);

% Verify that a read function exists
if (isempty(fmt_s.read))
    error('MATLAB:imread:readFunctionRegistration', 'No reading function for
format %s. See "help imformats".', ...
        fmt_s.ext{1});
end

if ((fmt_s.alpha) && (nargout == 3))

    % Use the alpha channel.
    [X, map, alpha] = feval(fmt_s.read, filename, extraArgs{:});

else

    % Alpha channel is not requested or is not applicable.
    alpha = [];
    [X, map] = feval(fmt_s.read, filename, extraArgs{:});

end

% Delete temporary file from Internet download.
if (url)
    delete_download(filename);
end

%%%
%%% Function delete_download
%%%
function delete_download(filename)

try
    delete(filename);
catch
    warning('MATLAB:imread:tempFileDelete', 'Can"t delete temporary file
"%s".', filename)
end

```

```

%%
%% Function parse_inputs
%%
function [filename, format, extraArgs, msg] = ...
    parse_inputs(varargin)

filename = "";
format = "";
extraArgs = {};
msg = "";

% Parse arguments based on their number.
switch(nargin)
case 0

    % Not allowed.
    msg = 'Too few input arguments.';
    return;

case 1

    % Filename only.
    filename = varargin{1};

otherwise

    % Filename and format or other arguments.
    filename = varargin{1};

    % Check whether second argument is a format.
    if (ischar(varargin{2}))
        fmt_s = imformats(varargin{2});
    else
        fmt_s = struct([]);
    end

    if (~isempty(fmt_s))
        % The argument matches a format.
        format = varargin{2};
        extraArgs = varargin(3:end);
    else
        % The argument begins the format-specific parameters.
        extraArgs = varargin(2:end);
    end
end

```

end

```
function h=imshow(varargin)
%IMSHOW Display image.
% IMSHOW(I,N) displays the intensity image I with N discrete
% levels of gray. If you omit N, IMSHOW uses 256 gray levels on
% 24-bit displays, or 64 gray levels on other systems.
%
% IMSHOW(I,[LOW HIGH]) displays I as a grayscale intensity
% image, specifying the data range for I. The value LOW (and
% any value less than LOW) displays as black, the value HIGH
% (and any value greater than HIGH) displays as white, and
% values in between display as intermediate shades of
% gray. IMSHOW uses the default number of gray levels. If you
% use an empty matrix ([]) for [LOW HIGH], IMSHOW uses
% [min(I(:)) max(I(:))]; the minimum value in I displays as
% black, and the maximum value displays as white.
%
% IMSHOW(BW) displays the binary image BW. Values of 0 display
% as black, and values of 1 display as white.
%
% IMSHOW(X,MAP) displays the indexed image X with the colormap
% MAP.
%
% IMSHOW(RGB) displays the truecolor image RGB.
%
% IMSHOW(...,DISPLAY_OPTION) displays the image, calling
% TRUESIZE if DISPLAY_OPTION is 'truesize', or suppressing the
% call to TRUESIZE if DISPLAY_OPTION is 'notruesize'. Either
% option string can be abbreviated. If you do not supply this
% argument, IMSHOW determines whether to call TRUESIZE based on
% the setting of the 'ImshowTruesize' preference.
%
% IMSHOW(x,y,A,...) uses the 2-element vectors x and y to
% establish a nondefault spatial coordinate system, by
% specifying the image XData and YData. Note that x and y can
% have more than 2 elements, but only the first and last
% elements are actually used.
%
% IMSHOW(FILENAME) displays the image stored in the graphics
% file FILENAME. IMSHOW calls IMREAD to read the image from the
% file, but the image data is not stored in the MATLAB
% workspace. The file must be in the current directory or on
% the MATLAB path.
```

```

%
% H = IMSHOW(...) returns the handle to the image object
% created by IMSHOW.
%
% Class Support
% -----
% The input image can be of class logical, uint8, uint16,
% or double, and it must be nonsparse.
%
% Remarks
% -----
% You can use the IPTSETPREF function to set several toolbox
% preferences that modify the behavior of IMSHOW:
%
% - 'ImshowBorder' controls whether IMSHOW displays the image
%   with a border around it.
%
% - 'ImshowAxesVisible' controls whether IMSHOW displays the
%   image with the axes box and tick labels.
%
% - 'ImshowTruesize' controls whether IMSHOW calls the TRUESIZE
%   function.
%
% For more information about these preferences, see the
% reference entry for IPTSETPREF.
%
% See also IMREAD, IMVIEW, IPTGETPREF, IPTSETPREF, SUBIMAGE,
% TRUESIZE, WARP, IMAGE, IMAGESC.

% Copyright 1993-2003 The MathWorks, Inc.
% $Revision: 5.37.4.6 $ $Date: 2003/08/23 05:52:43 $

% 1. Parse input arguments
% 2. Get an axes to plot in.
% 3. Create the image and axes objects and set their display
%   properties.
% 4. If the image is alone in the figure, position the axes
%   according to the current IMBORDER setting and then call
%   TRUESIZE.

newFigure = isempty(get(0,'CurrentFigure')) | ...
    strcmp(get(0,'CurrentFigure'),'NextPlot'),'new');

[cdata, cdatamapping, clim, map, xdata, ydata, filename, ...
    truesizeStr] = ParseInputs(varargin{:});

```

```

if (newFigure)
    figHandle = figure('Visible', 'off');
    axHandle = axes('Parent', figHandle);
else
    axHandle = newplot;
    figHandle = get(axHandle, 'Parent');
end

% Make the image object.
hh = image(xdata, ydata, cdata, 'BusyAction', 'cancel', ...
    'Parent', axHandle, 'CDataMapping', cdatamapping, ...
    'Interruptible', 'off');

% Set axes and figure properties if necessary to display the
% image object correctly.
showAxes = iptgetpref('ImshowAxesVisible');
set(axHandle, ...
    'TickDir', 'out', ...
    'XGrid', 'off', ...
    'YGrid', 'off', ...
    'DataAspectRatio', [1 1 1], ...
    'PlotBoxAspectRatioMode', 'auto', ...
    'Visible', showAxes);
set(get(axHandle, 'Title'), 'Visible', 'on');
set(get(axHandle, 'XLabel'), 'Visible', 'on');
set(get(axHandle, 'YLabel'), 'Visible', 'on');
if (~isempty(map))
    set(figHandle, 'Colormap', map);
end
if (~isempty(clim))
    set(axHandle, 'CLim', clim);
end

% Do truesize if called for.
truesizePref = iptgetpref('ImshowTruesize');
autoTruesize = strcmp(truesizePref, 'auto');
singleImage = SingleImageDefaultPos(figHandle, axHandle);

% Syntax specification overrides truesize preference setting.
if (strcmp(truesizeStr, 'notruesize'))
    callTruesize = 0;
elseif (strcmp(truesizeStr, 'truesize'))
    callTruesize = 1;

```

```

else
    % If there was no command-line override, and the truesize preference
    % is 'on', we still might not want to call truesize if the image
    % isn't the only thing in the figure, or if it isn't in the
    % default position.

    if (autoTruesize)
        callTruesize = singleImage;
    else
        callTruesize = 0;
    end
end

% Adjust according to ImshowBorder setting, unless we don't have a single
% image in the default position.
borderPref = iptgetpref('ImshowBorder');
if (strcmp(borderPref, 'tight') && singleImage)
    % Have the image fill the figure.
    set(axHandle, 'Units', 'normalized', 'Position', [0 0 1 1]);

    % The next line is so that a subsequent plot(1:10) goes back
    % to the default axes position instead of taking up the
    % entire figure.
    set(figHandle, 'NextPlot', 'replacechildren');
end

if (callTruesize)
    truesize(figHandle);
end

if (~isempty(filename) && isempty(get(get(axHandle, 'Title'), 'String')))
    set(get(axHandle, 'Title'), 'String', filename, ...
        'Interpreter', 'none', ...
        'Visible', 'on');
end

if (nargout > 0)
    % Only return handle if caller requested it.
    h = hh;
end

if (newFigure)
    set(figHandle, 'Visible', 'on');
end

```

```

%-----
% Subfunction ParseInputs
%-----

function [cdata, cdatamapping, clim, map, xdata, ...
         ydata, filename, truesizeStr] = ParseInputs(varargin)

% Defaults
filename = "";
truesizeStr = "";
clim = []; % stays empty for indexed and RGB
map = [];
image_type = "";
N_gray_levels = [];
xdata = [];
ydata = [];

% If nargin > 1, see if there's a trailing string argument.
if ((nargin > 1) && (ischar(varargin{end})))
    str = varargin{end};
    varargin(end) = []; % remove string from input arg list
    str = {'truesize', 'notruesize'};
    idx = strmatch(str, str);
    if (isempty(idx))
        eid = 'Images:imshow:unrecognizedOption';
        error(eid, 'Unknown option string "%s"', str);
    elseif (length(idx) > 1)
        eid = 'Images:imshow:ambiguousOption';
        error(eid, 'Ambiguous option string "%s"', str);
    else
        truesizeStr = str{idx};
    end
end

switch length(varargin)
case 0
    eid = 'Images:imshow:tooFewInputs';
    error(eid, '%s', 'Not enough input arguments. See HELP IMSHOW');

case 1
    % IMSHOW(I)
    % IMSHOW(RGB)
    % IMSHOW(FILENAME)

    if (ischar(varargin{1}))
        % IMSHOW(FILENAME)

```

```

filename = varargin{1};
[cdata,map] = imread(filename);
if (isempty(map))
    if (ndims(cdata) ~= 3)
        image_type = 'intensity';
    else
        image_type = 'truecolor';
    end

else
    image_type = 'indexed';
end

elseif (ndims(varargin{1}) == 3)
    % IMSHOW(RGB)
    image_type = 'truecolor';
    cdata = varargin{1};

else
    % IMSHOW(I)
    image_type = 'intensity';
    cdata = varargin{1};

end

case 2
    % IMSHOW(I,N)
    % IMSHOW(I,[a b])
    % IMSHOW(X,map)
    % IMSHOW(I,[])

if (numel(varargin{2}) == 1)
    % IMSHOW(I,N)
    image_type = 'intensity';
    cdata = varargin{1};
    N_gray_levels = varargin{2};

elseif (isequal(size(varargin{2}), [1 2]))
    % IMSHOW(I,[a b])
    image_type = 'intensity';
    cdata = varargin{1};
    clim = varargin{2};

elseif (size(varargin{2},2) == 3)
    % IMSHOW(X,map)
    image_type = 'indexed';

```

```

        cdata = varargin{1};
        map = varargin{2};

elseif (isempty(varargin{2}))
    % IMSHOW(I,[])
    image_type = 'intensity';
    cdata = varargin{1};
    clim = [min(cdata(:)) max(cdata(:))];

else
    eid = 'Images:imshow:invalidInputs';
    error(eid, '%s', 'Invalid input arguments; see HELP IMSHOW');

end

case 3
    % IMSHOW(x,y,RGB)
    % IMSHOW(x,y,I)
    % IMSHOW(R,G,B) OBSOLETE

    if (ndims(varargin{3}) == 3)
        % IMSHOW(x,y,RGB)
        image_type = 'truecolor';
        cdata = varargin{3};

        xdata = varargin{1};
        ydata = varargin{2};

    elseif (isvector(varargin{1}) && isvector(varargin{2}))
        % IMSHOW(x,y,I)
        image_type = 'intensity';
        cdata = varargin{3};

        xdata = varargin{1};
        ydata = varargin{2};

    elseif (isequal(size(varargin{1}),size(varargin{2}),size(varargin{3})),
        % IMSHOW(R,G,B)
        wid = 'Images:imshow:obsoleteSyntax';
        warning(wid, '%s', ['IMSHOW(R,G,B) is an obsolete syntax. ',...
        'Use a three-dimensional array to represent RGB image.']);
        image_type = 'truecolor';
        cdata = cat(3,varargin{:});

    else
        eid = 'Images:imshow:invalidType';

```

```

        error(eid, '%s', 'Invalid input arguments; see HELP IMSHOW');

    end

case 4
    % IMSHOW(x,y,I,N)
    % IMSHOW(x,y,I,[a b])
    % IMSHOW(x,y,X,MAP)
    % IMSHOW(x,y,I,[])

    if (numel(varargin{4}) == 1)
        % IMSHOW(x,y,I,N)
        image_type = 'intensity';
        cdata = varargin{3};
        N_gray_levels = varargin{4};

    elseif (isequal(size(varargin{4}), [1 2]))
        % IMSHOW(x,y,I,[a b])
        image_type = 'intensity';
        cdata = varargin{3};
        clim = varargin{4};

    elseif (size(varargin{4},2) == 3)
        % IMSHOW(x,y,X,map)
        image_type = 'indexed';
        cdata = varargin{3};
        map = varargin{4};

    elseif (isempty(varargin{4}))
        % IMSHOW(x,y,I,[])
        image_type = 'intensity';
        cdata = varargin{3};
        clim = [min(cdata(:)) max(cdata(:))];

    else
        eid = 'Images:imshow:invalidInputs';
        error(eid, '%s', 'Invalid input arguments. See HELP IMSHOW');

    end

    xdata = varargin{1};
    ydata = varargin{2};

case 5
    % IMSHOW(x,y,R,G,B) OBSOLETE
    wid = 'Images:imshow:obsoleteSyntax';

```

```

warning(wid, '%s', ['IMSHOW(x,y,R,G,B) is an obsolete syntax. ',...
'Use a three-dimensional array to represent RGB image.']);
image_type = 'truecolor';
cdata = cat(3,varargin{3:5});

xdata = varargin{1};
ydata = varargin{2};

otherwise
    eid = 'Images:imshow:tooManyInputs';
    error(eid, '%s', 'Too many input arguments. See HELP IMSHOW');

end

cdata = ValidateCdata(cdata);

if strcmp(image_type,'intensity') && strcmp(class(cdata),'logical')
    image_type = 'binary';
end

cdatamapping = GetCdataMapping(image_type);

if strcmp(cdatamapping,'scaled')
    map = GetMap(N_gray_levels);
    if isempty(clim)
        clim = GetClimFromCdataClass(cdata);
    end
end

end

if isempty(xdata) && isempty(ydata)
    xdata = [1 size(cdata,2)];
    ydata = [1 size(cdata,1)];
end

% Catch imshow(...,[ ]) case where input is constant.
if (~isempty(clim) && (clim(1) == clim(2)))
    % Do the Handle Graphics thing --- make the range [k-1 k+1].
    % Image will display as shade of gray.
    clim = double(clim) + [-1 1];
end

%%%
%%% -----
%%%
function [cdatamapping] = GetCdataMapping(image_type)

```

```

cdatamapping = 'direct';

% May want to treat binary images as 'direct'-indexed images for display
% in HG which requires no map.
%
% For now, they are treated as 'scaled'-indexed images for display in HG.

switch image_type
case {'intensity','binary'}
    cdatamapping = 'scaled';

case 'indexed'
    cdatamapping = 'direct';

end

%%%
%%% -----
%%%
function [map] = GetMap(N_gray_levels)

if isempty(N_gray_levels)
    if (get(0,'ScreenDepth') > 16)
        N_gray_levels = 256;
    else
        N_gray_levels = 64;
    end
end
map = gray(N_gray_levels);

%%%
%%% -----
%%%
function clim = GetClimFromCdataClass(cdata)

key_class = class(cdata);

switch key_class
case 'uint8'
    clim = [0 255];

case 'uint16'
    clim = [0 65535];

case {'logical','double'}

```

```

        clim = [0 1];

    otherwise
        eid = 'Images:imshow:invalidType';
        error(eid, '%s', 'Unsupported image class');

    end

    %%%
    %%% -----
    %%%
    function cdata = ValidateCdata(cdata)

    if ((ndims(cdata) > 3) || ((size(cdata,3) ~= 1) && (size(cdata,3) ~= 3)))
        eid = 'Images:imshow:unsupportedDimension';
        error(eid, '%s', 'Unsupported dimension')
    end

    if islogical(cdata) && (ndims(cdata) > 2)
        eid = 'Images:imshow:expected2D';
        error(eid, '%s', 'If input is logical (binary), it must be two-dimensional.');
```

end

```

    % RGB images can be only uint8, uint16, or double
    if ( (ndims(cdata) == 3) && ...
        ~isa(cdata, 'double') && ...
        ~isa(cdata, 'uint8') && ...
        ~isa(cdata, 'uint16') )
        eid = 'Images:imshow:invalidRGBClass';
        error(eid, 'RGB images must be uint8, uint16, or double.');
```

end

```

    % Clip double RGB images to [0 1] range
    if ( (ndims(cdata) == 3) && isa(cdata, 'double') )
        cdata(cdata > 1) = 1;
        cdata(cdata < 0) = 0;
    end

    % Catch complex CData case
    if (~isreal(cdata))
        wid = 'Images:imshow:displayingRealPart';
        warning(wid, '%s', 'Displaying real part of complex input');
        cdata = real(cdata);
    end
end

```

```

%%
%% Subfunction SingleImageDefaultPos
%%
function tf = SingleImageDefaultPos(figHandle, axHandle)

if (length(findobj(axHandle, 'Type', 'image')) > 1)
    % More than one image in axes
    tf = 0;

else

    figKids = allchild(figHandle);
    kids = [findobj(figKids, 'Type', 'axes') ;
            findobj(figKids, 'Type', 'uicontrol', 'Visible', 'on')];
    if (length(kids) > 1)
        % The axes isn't the only thing around, so don't truesize
        tf = 0;
    else
        % Is axHandle in the default position?
        activeProperty = get(axHandle, 'ActivePositionProperty');
        if (isequal(get(axHandle, activeProperty), ...
                    get(get(axHandle, 'Parent'), ['DefaultAxes' activeProperty])))
            % Yes, call truesize
            tf = 1;

        else
            % No, don't call truesize
            tf = 0;
        end
    end
end

end

%FIGURE Create figure window.
% FIGURE, by itself, creates a new figure window, and returns
% its handle.
%
% FIGURE(H) makes H the current figure, forces it to become visible,
% and raises it above all other figures on the screen. If Figure H
% does not exist, and H is an integer, a new figure is created with
% handle H.
%
% GCF returns the handle to the current figure.
%
% Execute GET(H) to see a list of figure properties and
% their current values. Execute SET(H) to see a list of figure

```

```

% properties and their possible values.
%
% See also SUBPLOT, AXES, GCF, CLF.

% Copyright 1984-2002 The MathWorks, Inc.
% $Revision: 5.10 $ $Date: 2002/04/10 17:07:17 $
% Built-in function.

```

```

%DOUBLE Convert to double precision.
% DOUBLE(X) returns the double precision value for X.
% If X is already a double precision array, DOUBLE has
% no effect.
%
% DOUBLE is called for the expressions in FOR, IF, and WHILE loops
% if the expression isn't already double precision. DOUBLE should
% be overloaded for all objects where it makes sense to convert it
% into a double precision value.
%
% See also SINGLE, DATATYPES, ISFLOAT, ISNUMERIC.

```

```

% Copyright 1984-2004 The MathWorks, Inc.
% $Revision: 1.8.4.2 $ $Date: 2004/03/08 02:01:31 $

```

```

function f = fft2(x, mrows, ncols)
%FFT2 Two-dimensional discrete Fourier Transform.
% FFT2(X) returns the two-dimensional Fourier transform of matrix X.
% If X is a vector, the result will have the same orientation.
%
% FFT2(X,MROWS,NCOLS) pads matrix X with zeros to size MROWS-by-
NCOLS
% before transforming.
%
% Class support for input X:
% float: double, single
%
% See also FFT, FFTN, FFTSHIFT, FFTW, IFFT, IFFT2, IFFTN.

```

```

% Copyright 1984-2004 The MathWorks, Inc.
% $Revision: 5.13.4.2 $ $Date: 2004/03/09 16:16:17 $

```

```

if ndims(x)==2
    if nargin==1
        f = fftn(x);
    else

```

```

        f = fftn(x,[mrows ncols]);
    end
else
    if nargin==1
        f = fft(fft(x,[],2),[],1);
    else
        f = fft(fft(x,ncols,2),mrows,1);
    end
end
end

```

```

function x = ifft2(varargin)
%IFFT2 Two-dimensional inverse discrete Fourier transform.
% IFFT2(F) returns the two-dimensional inverse Fourier transform of matrix
% F. If F is a vector, the result will have the same orientation.
%
% IFFT2(F,MROWS,NCOLS) pads matrix F with zeros to size MROWS-by-
% NCOLS
% before transforming.
%
% IFFT2(..., 'symmetric') causes IFFT2 to treat F as conjugate symmetric
% in two dimensions so that the output is purely real. This option is
% useful when F is not exactly conjugate symmetric merely because of
% round-off error. See the reference page for the specific mathematical
% definition of this symmetry.
%
% IFFT2(..., 'nonsymmetric') causes IFFT2 to make no assumptions about the
% symmetry of F.
%
% Class support for input F:
% float: double, single
%
% See also FFT, FFT2, FFTN, FFTSHIFT, FFTW, IFFT, IFFN.

% Copyright 1984-2004 The MathWorks, Inc.
% $Revision: 5.16.4.3 $ $Date: 2004/03/09 16:16:24 $

```

```

error(nargchk(1, 4, nargin, 'struct'))

```

```

f = varargin{1};
m_in = size(f, 1);
n_in = size(f, 2);
num_inputs = numel(varargin);
symmetry = 'nonsymmetric';
if ischar(varargin{end})
    symmetry = varargin{end};
    num_inputs = num_inputs - 1;

```

```

end

if num_inputs == 1
    m_out = m_in;
    n_out = n_in;

elseif num_inputs == 2
    error('MATLAB:ifft2:invalidSyntax', ...
        'If you specify MROWS, you also have to specify NCOLS.')

```

```

% See also FLIPUD, FLIPLR, FLIPDIM.

% From John de Pillis 19 June 1985
% Modified 12-19-91, LS.
% Copyright 1984-2003 The MathWorks, Inc.
% $Revision: 5.11.4.1 $ $Date: 2003/05/01 20:41:57 $

if ndims(A)~=2
    error('MATLAB:rot90:SizeA', 'A must be a 2-D matrix.');
```

end

```

[m,n] = size(A);
if nargin == 1
    k = 1;
else
    if length(k)~=1
        error('MATLAB:rot90:kNonScalar', 'k must be a scalar.');
```

end

```

k = rem(k,4);
if k < 0
    k = k + 4;
end
end
if k == 1
    A = A.';
    B = A(n:-1:1,:);
elseif k == 2
    B = A(m:-1:1,n:-1:1);
elseif k == 3
    B = A(m:-1:1,:);
    B = B.';
else
    B = A;
End

function I = rgb2gray(varargin)
%RGB2GRAY Convert RGB image or colormap to grayscale.
% RGB2GRAY converts RGB images to grayscale by eliminating the
% hue and saturation information while retaining the
% luminance.
%
% I = RGB2GRAY(RGB) converts the truecolor image RGB to the
% grayscale intensity image I.
%
% NEWMAP = RGB2GRAY(MAP) returns a grayscale colormap
% equivalent to MAP.
%
```

```

% Class Support
% -----
% If the input is an RGB image, it can be of class uint8,
% uint16 or double; the output image I is of the same class
% as the input image. If the input is a colormap, the input
% and output colormaps are both of class double.
%
% Example
% -----
% I = imread('board.tif');
% J = rgb2gray(I);
% imview(I), imview(J);
%
% [X,map] = imread('trees.tif');
% gmap = rgb2gray(map);
% imview(X,map), imview(X,gmap);
%
% See also IND2GRAY, NTSC2RGB, RGB2IND, RGB2NTSC.

% Copyright 1993-2003 The MathWorks, Inc.
% $Revision: 5.20.4.3 $ $Date: 2003/08/23 05:54:35 $

X = parse_inputs(varargin{:});
origSize = size(X);

% Determine if input includes a 3-D array
threeD = (ndims(X)==3);

% Calculate transformation matrix
T = inv([1.0 0.956 0.621; 1.0 -0.272 -0.647; 1.0 -1.106 1.703]);
coef = T(1,:);

if threeD
    %RGB
    % Shape input matrix so that it is a n x 3 array and initialize output
    % matrix
    X = reshape(X(:),origSize(1)*origSize(2),3);
    sizeOutput = [origSize(1), origSize(2)];

    % Do transformation
    if isa(X, 'double')
        I = X*coef;
        I = min(max(I,0),1);
    else
        %uint8 or uint16
        I = imlincomb(coef(1),X(:,1),coef(2),X(:,2),coef(3),X(:,3), ...

```

```

        class(X));
    end
    %Make sure that the output matrix has the right size
    I = reshape(I,sizeOutput);

else
    % For backward compatability, this function handles uint8 and uint16
    % colormaps. This usage will be removed in a future release.
    if ~isa(X,'double')
        I = imlincomb(coef(1),X(:,1),coef(2),X(:,2),coef(3),X(:,3), class(X));
    else
        I = X*coef;
        I = min(max(I,0),1);
    end
    I = im2double(I);
    I = [I,I,I];
end

%%
%%Parse Inputs
%%
function X = parse_inputs(varargin)

checknargin(1,3,nargin,mfilename);

switch nargin
case 1
    if ndims(varargin{1})==2
        if (size(varargin{1},2) ~=3 || size(varargin{1},1) < 1)
            eid = sprintf('Images:%s:invalidSizeForColormap',mfilename);
            msg = 'MAP must be a m x 3 array.';
            error(eid,'%s',msg);
        end
        if ~isa(varargin{1},'double')
            wid = sprintf('Images:%s:notAValidColormap',mfilename);
            msg1 = 'MAP should be a double m x 3 array with values in the';
            msg2 = ' range [0,1].Convert your map to double using IM2DOUBLE.';
            warning(wid,'%s %s',msg1,msg2);
        end
    elseif (ndims(varargin{1})==3)
        if ((size(varargin{1},3) ~= 3))
            eid = sprintf('Images:%s:invalidInputSize',mfilename);
            msg = 'RGB must be a m x n x 3 array.';
            error(eid,'%s',msg);
        end
end

```

```

else
    eid = sprintf('Images:%s:invalidInputSize',mfilename);
    msg1 = 'RGB2GRAY only accepts a 2-D input for MAP or a 3-D input for ';
    msg2 = 'RGB.';
    error(eid,'%s %s',msg1,msg2);
end
X = varargin{1};

case 2
eid = sprintf('Images:%s:invalidNumberOfArguments',mfilename);
msg = 'Two input arguments are not allowed.';
error(eid,'%s',msg);

case 3
wid = sprintf('Images:%s:obsoleteSyntax',mfilename);
msg1 = 'RGB2GRAY(R,G,B) is an obsolete syntax.';
msg2 = 'Use a three-dimensional array to represent RGB image.';
warning(wid,'%s %s',msg1,msg2);
if ( any(size(varargin{1})~=size(varargin{2})) || any(size(varargin{1})~= ...
            size(varargin{3})) )
    eid = sprintf('Images:%s:inputsDontHaveSameSize',mfilename);
    msg = 'R, G, and B must all be the same size.';
    error(eid,'%s',msg);
end
X = cat(3,varargin{1},varargin{2},varargin{3});
end

%no logical arrays
if islogical(X)
    eid = sprintf('Images:%s:invalidType',mfilename);
    msg = 'RGB2GRAY does not accept logical arrays as inputs.';
    error(eid,'%s',msg);
end

function [varargout] = max(varargin)
%MAX   Largest component.
% For vectors, MAX(X) is the largest element in X. For matrices,
% MAX(X) is a row vector containing the maximum element from each
% column. For N-D arrays, MAX(X) operates along the first
% non-singleton dimension.
%
% [Y,I] = MAX(X) returns the indices of the maximum values in vector I.
% If the values along the first non-singleton dimension contain more
% than one maximal element, the index of the first one is returned.
%
% MAX(X,Y) returns an array the same size as X and Y with the

```

```

% largest elements taken from X or Y. Either one can be a scalar.
%
% [Y,I] = MAX(X,[],DIM) operates along the dimension DIM.
%
% When complex, the magnitude MAX(ABS(X)) is used, and the angle
% ANGLE(X) is ignored. NaN's are ignored when computing the maximum.
%
% Example: If X = [2 8 4 then max(X,[],1) is [7 8 9],
%              7 3 9]
%
% max(X,[],2) is [8 and max(X,5) is [5 8 5
%              9], 7 5 9].
%
% See also MIN, MEDIAN, MEAN, SORT.

% Copyright 1984-2003 The MathWorks, Inc.
% $Revision: 5.16.4.2 $ $Date: 2004/04/16 22:04:48 $

% Built-in function.

if nargout == 0
    builtin('max', varargin{:});
else
    [varargout{1:nargout}] = builtin('max', varargin{:});
End

```