

## Displaying Different Image Types

### On this page...

[Displaying Indexed Images](#)

[Displaying Grayscale Images](#)

[Displaying Binary Images](#)

[Displaying Truecolor Images](#)

If you need help determining what type of image you are working with, see [Image Types in the Toolbox](#).

### Displaying Indexed Images

To display an indexed image, using either `imshow` or `imtool`, specify both the image matrix and the colormap. This documentation uses the variable name `X` to represent an indexed image in the workspace, and `map` to represent the colormap.

```
imshow(X,map)
```

or

```
imtool(X,map)
```

For each pixel in `X`, these functions display the color stored in the corresponding row of `map`. If the image matrix data is of class `double`, the value 1 points to the first row in the colormap, the value 2 points to the second row, and so on. However, if the image matrix data is of class `uint8` or `uint16`, the value 0 (zero) points to the first row in the colormap, the value 1 points to the second row, and so on. This offset is handled automatically by the `imtool` and `imshow` functions.

If the colormap contains a greater number of colors than the image, the functions ignore the extra colors in the colormap. If the colormap contains fewer colors than the image requires, the functions set all image pixels over the limits of the colormap's capacity to the last color in the colormap. For example, if an image of class `uint8` contains 256 colors, and you display it with a colormap that contains only 16 colors, all pixels with a value of 15 or higher are displayed with the last color in the colormap.

[▲ Back to Top](#)

### Displaying Grayscale Images

To display a grayscale image, using either `imshow` or `imtool`, specify the image matrix as an argument. This documentation uses the variable name `I` to represent a grayscale image in the workspace.

```
imshow(I)
```

or

```
imtool(I)
```

Both functions display the image by *scaling* the intensity values to serve as indices into a grayscale colormap.

If `I` is `double`, a pixel value of 0.0 is displayed as black, a pixel value of 1.0 is displayed as white, and pixel values in between are displayed as shades of gray. If `I` is `uint8`, then a pixel value of 255 is displayed as white. If `I` is `uint16`, then a pixel value of 65535 is displayed as white.

Grayscale images are similar to indexed images in that each uses an m-by-3 RGB colormap, but you normally do not specify a colormap for a grayscale image. MATLAB displays grayscale images by using a grayscale system colormap (where  $R=G=B$ ). By default, the number of levels of gray in the colormap is 256 on systems with 24-bit color, and 64 or 32 on other systems. (See [Displaying Colors](#) for a detailed explanation.)

### Displaying Grayscale Images That Have Unconventional Ranges

In some cases, the image data you want to display as a grayscale image could have a display range that is outside the conventional toolbox range (i.e.,  $[0,1]$  for `single` or `double` arrays,  $[0,255]$  for `uint8` arrays,  $[0,65535]$  for `uint16` arrays, or  $[-32767,32768]$  for `int16` arrays). For example, if you filter a grayscale image, some of the output data could fall outside the range of the original data.

To display unconventional range data as an image, you can specify the display range directly, using this syntax for both the `imshow` and `imtool` functions.

```
imshow(I, 'DisplayRange', [low high])
```

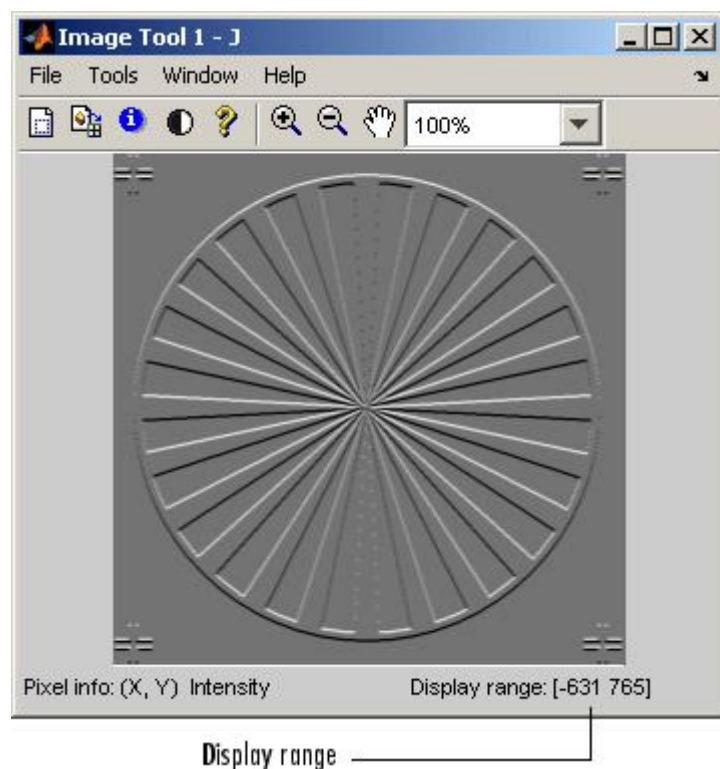
or

```
imtool(I, 'DisplayRange', [low high])
```

If you use an empty matrix (`[]`) for the display range, these functions scale the data automatically, setting `low` and `high` to the minimum and maximum values in the array.

The next example filters a grayscale image, creating unconventional range data. The example calls `imtool` to display the image, using the automatic scaling option. If you execute this example, note the display range specified in the lower right corner of the Image Tool window.

```
I = imread('testpat1.png');
J = filter2([1 2;-1 -2],I);
imtool(J, 'DisplayRange', []);
```



[▲ Back to Top](#)

## Displaying Binary Images

In MATLAB, a binary image is of class `logical`. Binary images contain only 0's and 1's. Pixels with the value 0 are displayed as black; pixels with the value 1 are displayed as white.

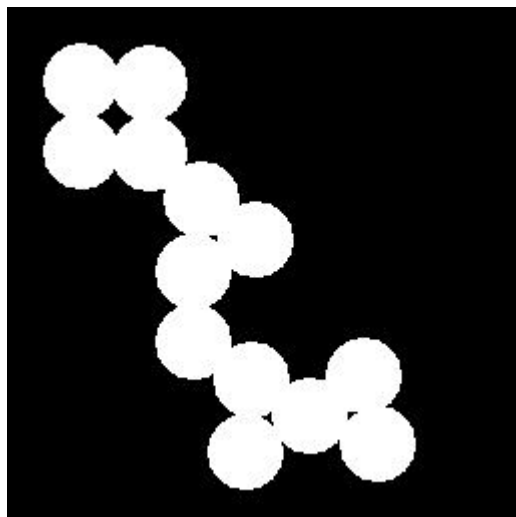
**Note** For the toolbox to interpret the image as binary, it must be of class `logical`. Grayscale images that happen to contain only 0's and 1's are not binary images.

To display a binary image, using either `imshow` or `imtool`, specify the image matrix as an argument. For example, this code reads a binary image into the MATLAB workspace and then displays the image. This documentation uses the variable name `BW` to represent a binary image in the workspace

```
BW = imread('circles.png');
imshow(BW)
```

or

```
imtool(BW)
```



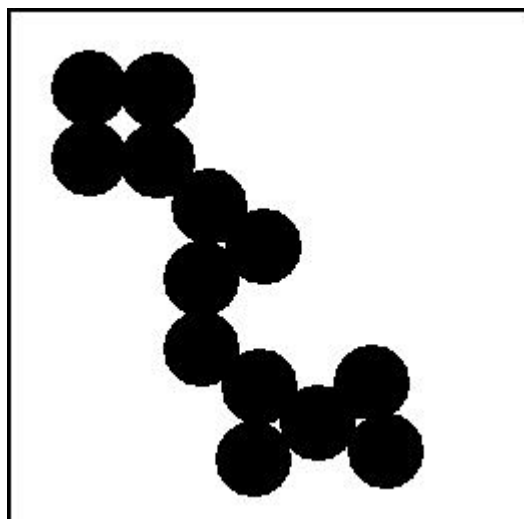
### Changing the Display Colors of a Binary Image

You might prefer to invert binary images when you display them, so that 0 values are displayed as white and 1 values are displayed as black. To do this, use the NOT (~) operator in MATLAB. (In this figure, a box is drawn around the image to show the image boundary.) For example:

```
imshow(~BW)
```

or

```
imtool(~BW)
```

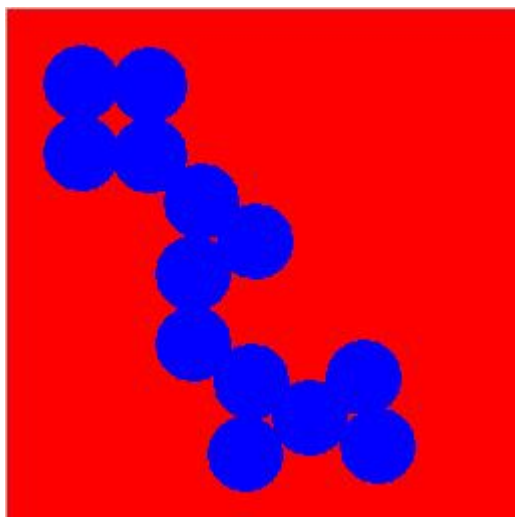


You can also display a binary image using the indexed image colormap syntax. For example, the following command specifies a two-row colormap that displays 0's as red and 1's as blue.

```
imshow(BW,[1 0 0; 0 0 1])
```

or

```
imtool(BW,[1 0 0; 0 0 1])
```



▲ [Back to Top](#)

## Displaying Truecolor Images

Truecolor images, also called RGB images, represent color values directly, rather than through a colormap. A truecolor image is an  $m$ -by- $n$ -by-3 array. For each pixel  $(x, c)$  in the image, the color is represented by the triplet  $(x, c, 1:3)$ .

To display a truecolor image, using either `imshow` or `imtool`, specify the image matrix as an argument. For example, this code reads a truecolor image into the MATLAB workspace and then displays the image. This documentation uses the variable name `RGB` to represent a truecolor image in the workspace

```
RGB = imread('peppers.png');
imshow(RGB)
```

or

```
imtool(RGB)
```




Systems that use 24 bits per screen pixel can display truecolor images directly, because they allocate 8 bits (256 levels) each to the red, green, and blue color planes. On systems with fewer colors, `imshow` displays the image using a combination of color approximation and dithering. See [Displaying Colors](#) for more information.

**Note** If you display a color image and it appears in black and white, check if the image is an indexed image. With indexed images, you must specify the colormap associated with the image. For more information, see [Displaying Indexed Images](#).

 [Back to Top](#)

[Provide feedback about this page](#)

 [Viewing Image Sequences](#)

[Adding a Colorbar to a Displayed Image](#) 

© 1984-2009 The MathWorks, Inc. • [Terms of Use](#) • [Patents](#) • [Trademarks](#) • [Acknowledgments](#)