Graphics

Changing Graph Data Rapidly

MATLAB plotting functions perform a wide variety of operations in the process of creating a graph to make plotting easier. For example, the `plot` function clears the current axes before drawing new lines, selects a line color or a marker type, searches for user-defined default values, and so on.

Low-Level Functions for Speed

The features that make plotting functions easy to use also consume computer resources. If you want to maximize graphing performance, you should use low-level functions and disable certain automatic features.

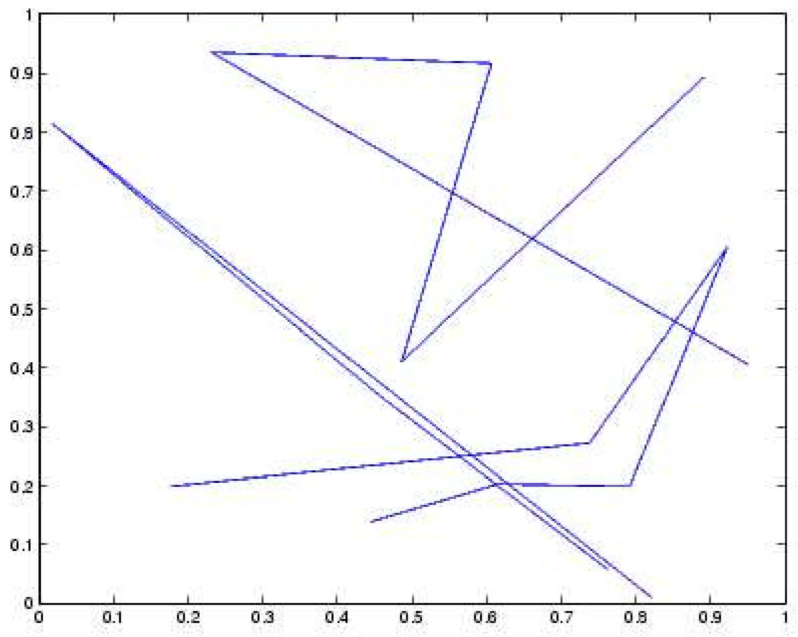
Low-level graphics functions (e.g., `line` vs. `plot`, surface vs. surf) perform fewer operation and therefore can be faster when you are creating many graphics objects. See High-Level Versus Low-Level for more information on how these functions differ.

Avoid Creating Graphics Objects

Each graphics object requires a certain amount of the computer's resources to create and store information, such as the value of all the object's properties. It is therefore more efficient to use fewer graphics objects if possible.

For example, you can add NaNs to vertex data (which causes that vertex to not be rendered) to create line segments that look like separate lines. You must place the NaNs at identical locations in each vector of data:

```
x = [rand(5,1);nan;rand(4,1);nan;rand(6,1)];
y = [rand(5,1);nan;rand(4,1);nan;rand(6,1)];
line(x,y);
```



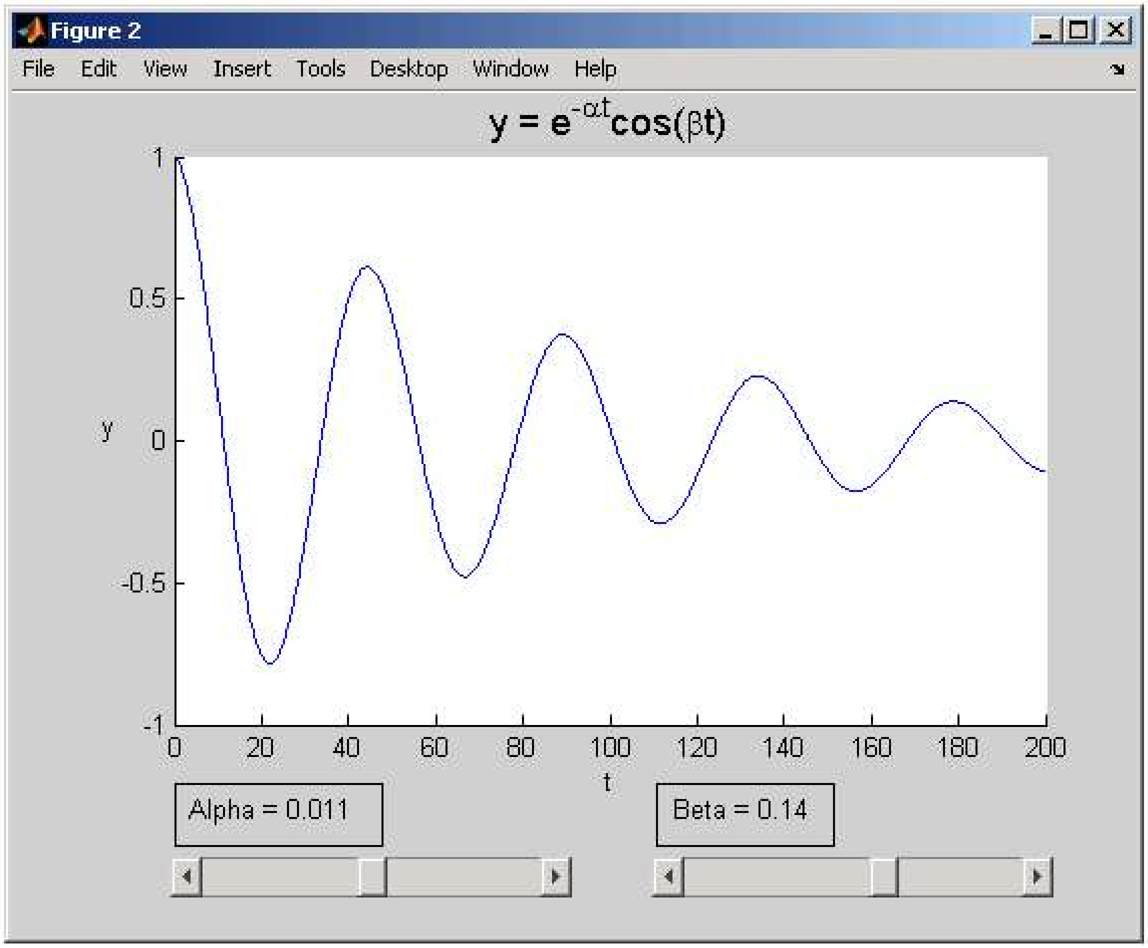
Update the Object's Data

If you want to view different data on what is basically the same graph, it is more efficient to update the data of the existing objects (lines, text, etc.) rather than recreating the entire graph.

For example, suppose you want to visualize the effect on your data of varying certain parameters. Follow these steps:

1. Set the limits of any axis that can be determined in advance (you can use `max` and `min` to determine the range of your data).
2. Recalculate the data using the new parameters.
3. Use the new data to update the data properties of the lines, text, etc. objects used in the graph.
4. Call `drawnow` to flush the event queue and update the figure (and all child objects in the figure).

The following example illustrates the use of these techniques in a GUI that uses sliders to vary two parameters in a mathematical expression, which is then plotted.



Note If you are using the MATLAB Help browser, you can run this example or open it in the MATLAB editor.

Performance of Bit-Mapped Images

Images can be defined with lower precision (than `double`) values to reduce the total amount of data required. MATLAB performs many operations on nondouble data types you can use smaller image data without converting the data to type `double`. See Working with 8-Bit and 16-Bit Images for more information.

Direct Color Mapping

Where possible, use indexed images because this type of image can apply direct mapping of pixel values to colormap values (`CDataMapping` set to `direct`). With direct mapping, MATLAB does not need to scale the data and then map it to values in the colormap.

See the `CDataMapping` image property for more information.

Use Truecolor for Smaller Images

The use of truecolor (red, green, and blue values) eliminates the need for color mapping. However, with very large images, the data can be quite large and thereby slow performance.

Direct Mapping of Transparency Values

If you are using an alphamap of transparency values, prescale the alpha data so you can use the most efficient alpha data mapping (`AlphaDataMapping` set to `none`)

See the `AlphaDataMapping` image property for more information.

Performance of Patch Objects

You can improve the speed with which MATLAB renders patch objects using the following techniques.

Define Patch Faces as Triangles

If you are using patch objects that have many vertices per patch face, you should modify your data so that each face has only three vertices, but still looks like your original object. This eliminates the tesselation step from the rendering process.

Use Data Thinning

It is sometimes possible (or even desirable) to reduce the number of vertices in a patch and still produce the desired results.

See the `reducepatch` and `reducevolume` functions for more information.

Direct Color Mapping

Where possible, use direct color mapping for coloring patches. (`CDataMapping` set to `direct`). With direct mapping, MATLAB does not need to scale the data and then map it to values in the colormap.

See the `CDataMapping` patch property for more information.

Use Truecolor for Smaller Patches

The use of truecolor (red, green, and blue values) eliminates the need for color mapping. However, with very large patches, the data can be quite large and thereby slow performance.

Direct Mapping of Transparency Values

If you are using an alphamap of transparency values, prescale the alpha data so you can use the most efficient alpha data mapping (`AlphaDataMapping` set to `none`)

See the `AlphaDataMapping` patch property for more information.

Performance of Surface Objects

You can improve the speed with which MATLAB renders surface objects using the following techniques.

Direct Color Mapping

Where possible, use direct color mapping for coloring surfaces. (`CDataMapping` set to `direct`). With direct mapping, MATLAB does not need to scale the data and then map it to values in the colormap.

See the `CDataMapping` surface property for more information.

Use Truecolor for Smaller Surfaces

The use of truecolor (red, green, and blue values) eliminates the need for color mapping. However, with very large surfaces, the data can be quite large and thereby slow performance.

Mapping of Transparency Values

If you are using an alphamap of transparency values, prescale the alpha data so you can use the most efficient alpha data mapping (`AlphaDataMapping` set to `none`)

See the `AlphaDataMapping` surface property for more information.

Use Texture-Mapped Face Color

If you are using surface objects in an animation or want to be able to pan and rotate them quickly, you can achieve better rendering performance with large surfaces by setting `EdgeColor` to `none` and `FaceColor` to `texture`.

This technique is particularly useful if you want a high resolution surface without creating an objects whose data is large and therefore, very slow to transform. For example,

```
h1 = surf(peaks(1000));
shading interp
cd1 = get(h1,'CData');
surf(peaks(24),'FaceColor','Texture','EdgeColor','none',...
    'CData',cd1)
```

Disable Automatic Modes   Figure Properties