matlab-graphics package reference

Greg Phillips, 20 September 2011

Contents

1 Overview

The matlab-graphics package is a simple, object-oriented graphics library for MATLAB. It was inspired by John Zelle's graphics.py library but is quite different due to differing goals and the fact that MATLAB Is Not Python.

In matlab-graphics, all drawing and interaction takes place on an area called "the drawing." You can add lines, rectangles, ellipses and text "shapes" to the drawing. You can change these shapes' sizes, angles and colours, you can move them around, and you can make them appear and disappear. You can wait for the user to click on the drawing and find out where the click happened. You can also create simple animations.

Here's a short example. It creates a fresh drawing and places a red rectangle with a narrow blue border, 100 units wide and 50 units high, at the drawing's bottom left corner. It asks the user to click somewhere, then moves the rectangle, in an animation lasting about one second, until the rectangle is centred where the user clicked. During the movement, the rectangle also completes one full rotation.

```
clc; clear;
import graphics.*

fresh;

STEPS = 25;
PAUSE_LENGTH = 1/STEPS;

rect = new_rectangle.width(100).height(50).edge_width(3).centre([50 25]);
rect.colour([1 0 0]).edge_colour([0 0 1]);

txt = new_text.string('Click to move the rectangle.').height(50);
end_point = get_point;
txt.visible(false);

move_step = (end_point - rect.centre)/STEPS;
rotation_step = (pi*2/STEPS);

for x = 1:STEPS
    rect.move(move_step).rotate(rotation_step);
    pause(PAUSE_LENGTH);
end
```

2 Installation and use

Download the current version of matlab-graphics from http://arity.ca/software/. Unzip it into the folder of your choice and change your MATLAB working folder to there. You should find a file called `click_to_move.m` (which is the example above) — try running it. If all is well, a window called “The Drawing” will open, you’ll be asked to click on the drawing, and when you do a short animation will run.

You will find a copy of the document you are currently reading in the `+graphics/doc/` folder.

Put your scripts in the folder where you found `click_to_move.m` (which you can delete if you like). To use the matlab-graphics functions and shapes, you must have the line:

```
import graphics.*
```

near the beginning of your script.

3 Key concepts

This section describes the key concepts of the matlab-graphics package. Subsequent sections provide references for each of the shape types.

3.1 The Drawing, points, and drawing units

The drawing is a square area, 1000 by 1000 “drawing units” in extent. There is only one drawing at a time in a matlab-graphics program. At the beginning of a matlab-graphics program you should always create a new drawing or erase the contents of the existing drawing by calling the `fresh` function, like this:

```
fresh;
```

The lower left corner of the drawing is the point `[0 0]` and the upper right is `[1000 1000]`. The first coordinate in a point is its horizontal distance from the origin, the second coordinate its vertical distance. Points are just MATLAB vectors containing two numbers.

All locations, displacements, widths and heights are specified in drawing units. Drawing units are real-valued and can have any precision. So, for example, a width of 32.5 is fine, as is the point [129.0721 936.58]. MATLAB takes care of mapping drawing units to screen pixels.

3.2 Shapes

matlab-graphics provides four shape types: `line`, `rectangle`, `ellipse`, and `text`. You can create as many of each of these shape types as you like. All shapes have a centre, a colour and an angle, and can be either visible or hidden. Each shape also has other properties, depending what it is. Lines have start points, end points, lengths and widths; text shapes have fonts, heights, and strings (the text itself); and ellipses and rectangles have edge widths and edge colours.

Shapes are created by calling shape creation functions and may have their properties queried and modified by invoking methods on them. The Shapes reference section provides a complete list and description of the shape creation and property-querying and -modifying methods available for each shape type. In this section we explain how these methods can be used in your programs.

If a shape extends past the edge of the drawing it will be “clipped” so the part that is “off drawing” is not shown. The exception is text shapes, which can extend past the edge of the drawing but which will *disappear completely* (including the parts that should be in the drawing area) if they extend too far.

This is a MATLAB quirk I would dearly love to fix!

3.2.1 Creating shapes

To create a shape you call the corresponding function from the `+graphics/` folder. For example, you create a new ellipse like this:

```
new_ellipse;
```

Each shape is created with a default set of properties. A new ellipse will be visible, 300 units wide and 200 high, a light brown colour with a darker brown edge 5 units wide, at an angle of 0, and centred on the point [500 500].

Of course, if you want to do anything with a shape later you need to assign it to a variable, like this:

```
e = new_ellipse;
```

or store it in a matrix, like this:

```
ellipselist = [ ];
% lots of other ellipses added here...
ellipselist = [ellipselist new_ellipse];
```

MATLAB won't let you mix different kinds of shapes in a single matrix. If you need to do that, you'll have to use a cell array.

3.2.2 Querying and modifying shape properties

A shape's properties can be queried by invoking methods on the shape. For instance, in this code fragment we create a new ellipse, name it e, then ask for e's width:

```
e = new_ellipse;
w = e.width;
```

See the second line? The code `e.width` means "invoke (execute) the `width` method of the shape referred to by e." Invoking the `width` method like this, without providing any parameters, will return the current value of e's `width` property. After running this code the value of the variable `w` will be 300, which is the default width for a new ellipse.

A shape's properties can be set by invoking the appropriate method and providing a parameter of the correct type. Continuing the example above, we can change e's width to 375.5 like this:

```
e.width(375.5);
```

Each method that modifies a shape's properties returns a reference to the shape itself. So if we do this:

```
f = e.height(55);
```

then the height of e will be set to 55 and `f` will refer to *the same ellipse* as e.

3.2.3 Method chaining

Returning a reference to the shape itself is a useful feature because it lets us do a trick called “method chaining.” This is where we invoke several methods in a row, like this:

```
e.height(99).width(132).angle(pi/4).centre([300 700]);
```

In the line above we first set e’s height to 99. The `height` method returns the ellipse it was called on, so we can then immediately invoke that ellipse’s `width` method, followed by its `angle` method, and finally its `centre` method. The line above is exactly equivalent to this:

```
e.height(99);
e.width(132);
e.angle(pi/4);
e.centre([300 700]);
```

... but quite a bit more compact! A warning though: if there’s an error on a method-chained line it may be difficult to tell which method call is causing it. If this happens, a good debugging technique is to split the chain into separate calls.

3.3 Colours

Colours in matlab-graphics are given as vectors containing three numbers between 0.0 and 1.0. The first number in the list indicates the amount of red in the colour, the second the amount of green, and the third the amount of blue. 0.0 means “none of this” and 1.0 means “the maximum amount,” so [0 0 0] (no colour at all) is black and [1 1 1] (full intensity of all the colours) is white. Any colour that can be displayed on screen can be specified this way.

Here are a few more examples:

- bright red: `[1 0 0]`
- darker red: `[0.5 0 0]`
- bright green: `[0 1 0]`
- bright blue: `[0 0 1]`
- bright yellow: `[1 1 0]`
- bright magenta: `[1 0 1]`
- bright cyan: `[0 1 1]`
- tangerine orange: `[1 0.5 0]`
- mocha brown: `[0.5 0.25 0]`
- eggplant purple: `[0.25 0 0.5]`
- medium grey: `[0.5 0.5 0.5]`

In matlab-graphics “`colour`” is spelled with a “u” because I’m Canadian and that’s the way we do things Around Here! However, “`color`” is an acceptable substitute everywhere.

3.4 User interaction (get_point)

matlab-graphics offers one mechanism for accepting user input: the `get_point` function. When called, it waits for the user to click on the drawing then returns the point on which the user clicked. For example if we execute the line:

```
p = get_point;
```

and the user clicks towards in the lower right of the drawing, `p` will be equal to something like [852.3392 156.4327] afterwards.

3.5 Animation

matlab-graphics provides a simple animation capability based on MATLAB's `pause` function. By default, pausing is enabled. If you want to disable it (for example, when testing a script), you can turn it off by calling the MATLAB `pause` function with `off` as a parameter:

```
pause off;
```

You can insert a pause into your program at any time by calling the `pause` function with a floating point parameter representing the desired pause time in seconds:

```
pause(0.2);
```

For example, to create a text object and then cause it to fade to white over a period of about two seconds, we can do this:

```
t = new_text.string('Help, I''m fading!');
for count = 1:50
    pause(0.04);
    t.colour(min([1 1 1], t.colour + 0.02));
end
```

The initial text appears almost immediately. By default, a new text shape is coloured [0 0 0] (black). In the for loop we fade the text to [1 1 1] (white) by adding 0.02 to its colour fifty times. The call to `min` ensures that rounding errors won't push the colour past [1 1 1], which would cause an error.

Calling `pause(0.04)` before each colour change causes MATLAB to wait about 0.04 seconds (one twenty-fifth of a second). This means that the fifty-step for loop will take about two seconds (0.04 ■ 50) to execute — actually, slightly more.

If you turn `pause off` and run this code again, the `pause(0.04)` will have no effect and the change from black to white will appear to be instant. In fact, you probably won't see the black text at all.

4 Shapes reference

4.1 All shapes

Each shape can be visible or hidden, and has a centre point, an angle, and a colour. Any of these properties can be changed by calling the appropriate method. For convenience, `move` and `rotate` methods are provided; these act on the centre point and the angle, respectively.

4.1.1 visible

Invoked with no arguments, returns the shape's current visibility, either `true` or `false`:

```
v = shape.visible;
```

Invoked with a boolean as argument, sets that as the shape's visibility and returns the shape:

```
shape.visible(false);
```

4.1.2 centre

Invoked with no arguments, returns the current centre point of the shape:

```
p = shape.centre;
```

Invoked with a point as argument, sets that as the shape's centre point and returns the shape:

```
shape.centre([400 700]);
```

4.1.3 move

Invoked with no arguments, returns [0 0]. You would not normally want to do this!

```
zeros = shape.move;
```

Invoked with a two number vector representing a displacement, adds that to the shape's centre point and returns the shape:

```
shape.move([10 20]);
```

Of course, the same effect can be achieved using the centre method:

```
shape.centre(shape.centre + [10 20]);
```

4.1.4 angle

Invoked with no arguments, returns the current angle of the shape in radians:

```
a = shape.angle;
```

Invoked with an angle in radians, sets that as the shape's angle and returns the shape. Rotation is about the shape's centre:

```
shape.angle(pi/4);
```

4.1.5 rotate

Invoked with no arguments, returns 0. You would not normally want to do this!

```
zero = shape.rotate;
```

Invoked with an angle in radians, rotates the shape by that angle and returns the shape. Rotation is about the shape's centre:

```
shape.rotate(pi/8);
```

Of course, the same effect can be achieved using the angle method:

```
shape.angle(shape.angle + pi/8);
```

4.1.6 colour (or color)

Invoked with no arguments, returns the current main colour of the shape, which will be a three-element vector with values between [0 0 0] and [1 1 1]. See Colours for more details:

```
c = shape.colour;
```

Invoked with a colour vector, sets the shape's main colour to that value and returns the shape:

```
shape.colour([0.25 0.5 0]);
```

matlab-graphics accepts "`color`" as a alternative spelling of "`colour`".

4.2 Ellipses and rectangles

Ellipses and rectangles offer exactly the same methods as one another. The only difference is that ellipses are curved and rectangles have corners. Go figure. An ellipse or rectangle has a height, a width, an edge width and an edge colour.

See also the methods `visible`, `centre`, `move`, `angle`, `rotate` and `colour` in All shapes. For ellipses and rectangles, `colour` sets the body colour; the edge colour can be set separately using `edge_colour`.

4.2.1 new_ellipse

Creates a new ellipse centred at [500 500], 300 wide, 200 high, with an edge width of 5, a colour of [0.5 0.3 0.3] (light brown) and an edge colour of [0.35 0.2 0.2] (darker brown) and an angle of zero:

```
e = new_ellipse;
```

4.2.2 new_rectangle

Creates a new rectangle centred at [500 500], 300 wide, 200 high, with an edge width of 5, a colour of [0.3 0.3 0.5] (light blue) and an edge colour of [0.2 0.2 0.35] (darker blue) and an angle of zero:

```
r = new_rectangle;
```

4.2.3 height

Invoked with no arguments, returns the current height of the ellipse or rectangle:

```
h = e.height;
```

Invoked with a number, sets the ellipse or rectangle's height to that number and returns the shape:

```
e.height(250);
```

4.2.4 width

Invoked with no arguments, returns the current width of the ellipse or rectangle:

```
w = r.width;
```

Invoked with a number, sets the ellipse or rectangle's width to that number and returns the shape:

```
r.width(250);
```

4.2.5 edge_width

Invoked with no arguments, returns the current edge (outline or border) width of the ellipse or rectangle:

```
ew = e.edge_width;
```

Invoked with a number, sets the ellipse or rectangle's edge width to that number and returns the shape. The edge is *inside* the width and height of the shape, so making the edge wider will eat into the middle of the shape, not make the shape larger:

```
e.edge_width(20);
```

4.2.6 edge_colour (or edge_color)

Invoked with no arguments, returns the current edge colour of the ellipse or rectangle, which will be a three-element vector with values between [0 0 0] and [1 1 1]. See *Colours* for more details:

```
c = r.edge_colour;
```

Invoked with a colour vector, sets the ellipse or rectangle's edge colour to that value and returns the shape:

```
r.edge_colour([0.25 0.5 0]);
```

matlab-graphics accepts "`edge_color`" as a alternative spelling of "`edge_colour`".

4.3 Lines

A line has a start point, end point, length and width. See also the methods `visible`, `centre`, `move`, `angle`, `rotate`, and `colour` in All shapes.

Moving a line's centre changes its start and end points but not its length or angle. Changing a line's angle changes its start and end points but not its length or centre.

4.3.1 new_line

Creates a horizontal (angle zero) line, 500 long and 10 wide, centred at [500 500] and coloured [0.2 0.35 0.2] (dark green). As you can calculate, the line's start point is [250 500] and its end point is [750 500]:

```
ln = new_line;
```

4.3.2 start

Invoked with no arguments, returns the current start point of the line:

```
s = ln.start;
```

Invoked with a point, changes the line's start point to that point. Changing a line's start point may affect its length, centre and angle:

```
ln.start([25 25]);
```

4.3.3 end

Invoked with no arguments, returns the current end point of the line:

```
e = ln.end;
```

Invoked with a point, changes the line's end point to that point and returns the line. Changing a line's end point may affect its length, centre and angle:

```
ln.end([925 925]);
```

4.3.4 length

Invoked with no arguments, returns the current length of the line:

```
lg = ln.length;
```

Invoked with a number, changes the line's length to match that number and returns the line. The line is extended or contracted around its centre. Changing a line's length may affect its start point and end point:

```
ln.length(275);
```

4.3.5 width

Invoked with no arguments, returns the current width of the line:

```
w = ln.width;
```

Invoked with a number, sets the line's width to that number and returns the line:

```
ln.width(50);
```

4.4 Text

Text shapes have a string (the text itself), a font, and a height. See also the methods `visible`, `centre`, `move`, `angle`, `rotate` and `colour` in All shapes.

A text shape does not have a width property! Calling `width` on a text shape will result in an error.

Text shapes can extend past the edges of the drawing area without being clipped. However, if a text shape becomes wider than the window that the drawing is in, it will disappear entirely, including the part that would otherwise appear in the drawing. This is a MATLAB quirk I haven't figured out how to address.

4.4.1 new_text

Creates a new text shape with the string “text”, set in the Helvetica font, centred at [500 500], coloured [0 0 0] (black), with a height of 100 and an angle of zero:

```
t = new_text;
```

4.4.2 string

Invoked with no arguments, returns the current string value of the text shape:

```
s = t.string;
```

Invoked with a string, sets the text’s string value to that string and returns the text shape:

```
t.string('Now I''m something new!');
```

4.4.3 height

Invoked with no arguments, returns the current height of the text shape:

```
h = t.height;
```

Invoked with a number, sets the text shape’s height to that number and returns the text shape:

```
t.height(75);
```

4.4.4 font

Invoked with no arguments, returns the current font being used to render the text shape:

```
f = t.font;
```

Invoked with the name of a font, sets the current font for the text shape and returns the text shape. The font must be one currently available on your system:

```
t.font('Times New Roman');
```

5 Copyright, license and author contact information