fig2pov documentation

# Summary

Convert MATLAB figures to ray-traced images using Povray as the rendering engine. If desired, add texture and other rendering elements not supported by MATLAB to your graphics objects.

# Introduction

MATLAB graphics uses openGL or Painters for rendering figures. The quality of the rendering is not as good as what can be provided by e.g ray tracing. Povray is a popular open-source ray tracing program with its own scripting language for describing scenes.

This function provides a simple way to convert MATLAB figures (or, to be exact, axes) to images rendered using ray tracing. It does so by generating a povray script based on the contents of the axes object. This script can then be executed in Povray to generate the final image.

# Usage

After you've generated an axes object with all the graphics elements you want, this figure can be converted to a Povray script by simply calling:

|  |
| --- |
| fig2pov(h\_axes, script\_name) |

With:

* h\_axes the handle to the axes object you want to convert (use gca for most recent axes)
* script\_name the name you would like for the generated script

If no arguments are provided, the default values are h\_axes = gca and script\_name = 'fig.pov'.

To see the rendered version, you will have to execute this script in Povray. The program is free and can be downloaded here:

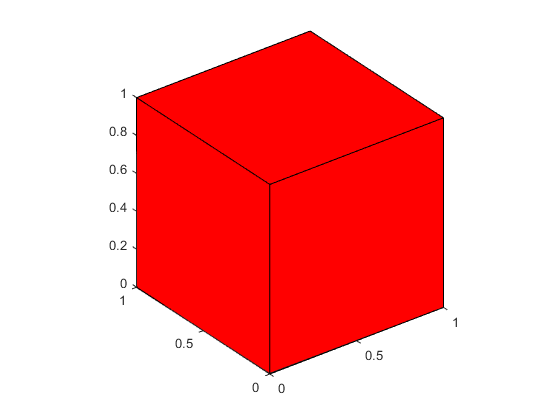
<https://www.povray.org/download/>

# A first example

Consider a red cube, drawn using the MATLAB patch object:

|  |
| --- |
| cube = patch('Vertices',[0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1], ...  'Faces',[1 2 4 3; 5 6 8 7; 1 2 6 5; 3 4 8 7; 1 3 7 5; 2 4 8 6], ...  'FaceColor', [1 0 0]);  view(3)  axis equal  h\_axes = gca; |

The code above will result in the following figure:



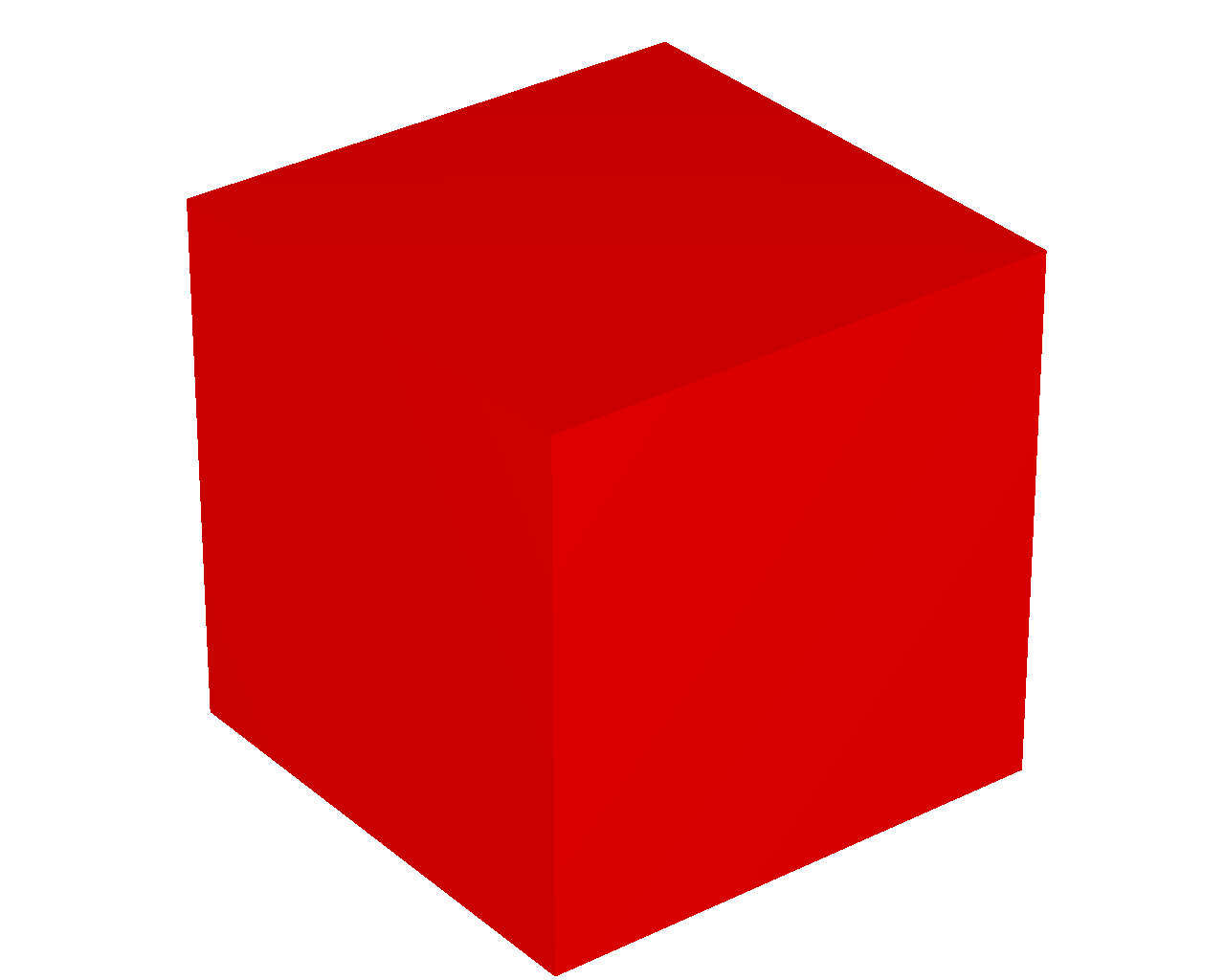
Next, call fig2pov to convert this figure into a Povray script:

|  |
| --- |
| fig2pov(gca, 'cube.pov'); |

The working directory will now contain a file called 'cube.pov', which is a file in the Povray scripting language. The \\*.pov format is readable and can be edited in any text editor. To render the image, the script has to be executed by Povray. Make sure Povray is installed and call it from either the command line or using the GUI version.

|  |
| --- |
| povray cube.pov |

In this case, the script will result in the following figure:



# Adding texture and other elements

In addition to a straightforward conversion from the MATLAB figure to a ray-traced image, it is possible to specify additional properties and options for the graphics objects. This will enable rendering options that are not natively supported in MATLAB itself, such as adding texture to patch objects, adding shadowing, etc ...

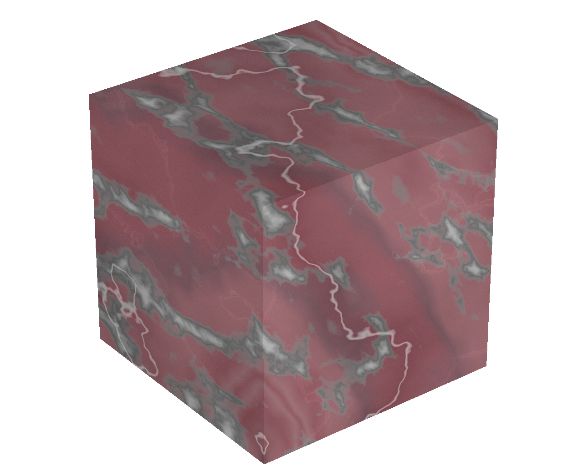
Each graphics object has a property called ‘UserData’, which allows any type of data to be attached to the object. The appearance a graphics object can be changed by adding a structure named 'povray' to the UserData of that object.

The appearance a graphics object can be further changed by adding a structure named 'povray' to the UserData of that object.

For instance, with 'cube' being the handle of the patch created earlier:

|  |
| --- |
| cube.UserData.povray.Texture = 'T\_Stone21'; |

When calling fig2pov, the same cube will now be rendered with a texture named 'T\_Stone21', giving the following figure:

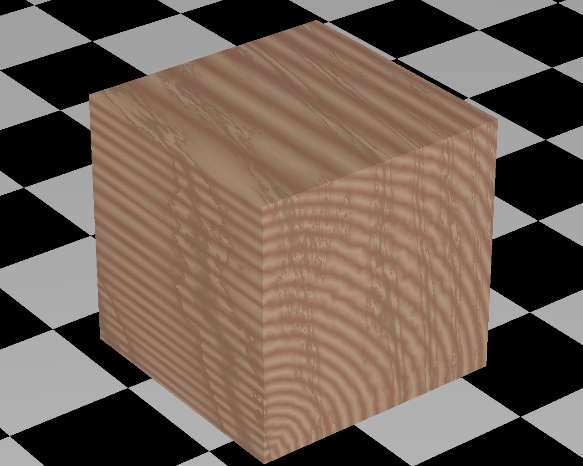


See [here](./Documents) for available textures.

Each graphics object can have its own 'UserData.povray' structure:

|  |
| --- |
| cube.UserData.povray.Texture = 'T\_Wood1';  h\_axes.UserData.povray.Plane = [0 0 1 -5];  h\_axes.UserData.povray.PlaneColor = 'checker color Black, color White'; |

This gives:



See the documentation (./Documents/fig2pov documentation.docx) for more details on the extra options that can be specified.

If an axes object contains multiple graphics objects, all of them will be included in the Povray script.

# Additional Povray options

Intro text.

## Additional Povray options for object of type ‘axes’

**Plane**

**PlaneColor**

**PlaneTexture**

**Define**

**Zoom**

Example:

|  |  |  |
| --- | --- | --- |
| cube = patch('Vertices',[0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1], ...  'Faces',[1 2 4 3; 5 6 8 7; 1 2 6 5; 3 4 8 7; 1 3 7 5; 2 4 8 6], ...  'FaceColor', [1 0 0]);  view(3)  axis equal  h\_axes = gca;  h\_axes.UserData.povray.Zoom = 0.5; | | |
|  |  |  |
| Zoom = 0.5 | Zoom = 1 | Zoom = 1.5 |

Default Value: No Zoom is applied (i.e. Zoom = 1)

## Additional Povray options for object of type ‘patch’

**Texture, InteriorTexture**

Example:

|  |  |
| --- | --- |
| cube = patch('Vertices',[0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1], ...  'Faces',[ 3 4 2 1; 5 6 8 7; 7 8 4 3; 5 7 3 1], ...  'FaceColor', [1 0 0]);  view(3)  axis equal  h\_axes = gca;  cube.UserData.povray.Texture = 'T\_Stone1';  cube.UserData.povray.InteriorTexture = 'T\_Stone18'; | |
|  |  |
| No InteriorTexture | InteriorTexture = 'T\_Stone18' |

Note:

* this depends on the correct orientation of the vertices in each face.

**TextureScale**

Example:

|  |  |  |
| --- | --- | --- |
| cube.UserData.povray.TextureScale = 1; | | |
|  |  |  |
| TextureScale = 1 | TextureScale = 5 | TextureScale = 0.2 |

Default Value: TextureScale = 1.

**ShadowLess**

**drawAsSphere, drawAsCylinder**

In MATLAB, patch and surface objects are really collections of polygons, even if the intention is to draw a sphere or a cylinder. The appearance of "roundness" is obtained by using large numbers of faces.

Povray natively supports commands that draw smooth spheres or cylinders (or any object with a rotation axis). If we know that an object is a sphere or has an symmetry axis, we can include this information in UserData.povray. fig2pov will then use the appropriate povray commands, rather than treating the object as a collection of faces.

**FaceColor**

**drawEdges**

**EdgeColor**

**EdgeTexture**

**MarkerTexture**

## Additional Povray options for object of type ‘surface’

**Texture, InteriorTexture**

See ‘Additional Povray options for object of type ‘patch’’

**TextureScale**

See ‘Additional Povray options for object of type ‘patch’’

**drawAsSphere**

Example:

|  |  |
| --- | --- |
| [x,y,z]=sphere;  h\_sphere = surf(x, y, z, 'FaceColor', 'r')  view(3)  axis equal  h\_sphere.UserData.povray.drawAsSphere = true; | |
|  |  |
| drawAsSphere = false | drawAsSphere = true |

Default Value: drawAsSphere = false

**drawAsCylinder**

|  |  |
| --- | --- |
| t = 0:pi/10:2\*pi;  [x, y, z] = cylinder(0.5+0.1\*cos(t));  h\_cyl = surf(x, y, z, 'FaceColor', 'r')  axis equal  h\_cyl.UserData.povray.drawAsCylinder = true; | |
|  |  |
| drawAsCylinder = false | drawAsCylinder = true |

**MeshOn**

Example:

|  |  |
| --- | --- |
| [X,Y,Z] = peaks(25);  h\_surf = surf(X, Y, Z/3);  view(3)  axis equal  h\_surf.UserData.povray.MeshOn = true; | |
|  |  |
| MeshOn= false | MeshOn=true |

Notes:

* in its current form, ‘MeshOn’ will only work or surfaces defined on an (X, Y)-grid. Not for e,g, spheres or cylinders.
* The color and thickness of the gridlines are controlled by the surface object properties ‘EdgeColor’ and ‘Linewidth’ respectively

**SmoothingOn**

Example:

|  |  |
| --- | --- |
| [X,Y,Z] = peaks(25);  h\_surf = surf(X, Y, Z/3);  view(3)  axis equal  h\_surf.UserData.povray.SmoothingOn = true;  h\_surf.FaceColor=[1 0 0]; | |
|  |  |
| SmoothingOn = false | SmoothingOn = true |

## Additional Povray options for object of type ‘line’

Povray is a ray-tracing program. As such, it has no concept of "lines". Instead, lines are converted into thin cylinders.

|  |  |
| --- | --- |
| h\_line=plot3(rand(10,1),rand(10,1),rand(10,1),'LineWidth',3);  view(3)  axis equal | |
|  |  |
| MATLAB | Povray |

**Texture, TextureScale**

|  |  |
| --- | --- |
| h\_line=plot3(rand(10,1), rand(10,1), rand(10,1),'LineWidth',5);  view(3)  axis equal  h\_line.UserData.povray.Texture = 'T\_Stone9'; | |
|  |  |
| Texture = 'T\_Stone9'  TextureScale = 1 | Texture = 'T\_Stone9'  TextureScale = 0.1 |

**SmoothingOn**

|  |  |
| --- | --- |
| h\_line=plot3(rand(10,1),rand(10,1),rand(10,1),'LineWidth',3);  view(3)  axis equal  h\_line.UserData.povray.SmoothingOn = true | |
|  |  |
| SmoothingOn = false | SmoothingOn = true |

## Additional Povray options for object of type ‘light’

**ShadowLess**

|  |  |
| --- | --- |
| [X,Y,Z] = peaks(25);  h\_surf = surf(X, Y, Z/3);  [x,y,z] = sphere;  h\_sphere=surf(x+2,y-2,z+3);  h\_light = camlight(‘right’);  view(3)  axis equal  h\_light.UserData.povray.Shadowless = false; | |
|  |  |
| Shadowless = false | Shadowless = true |