

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:

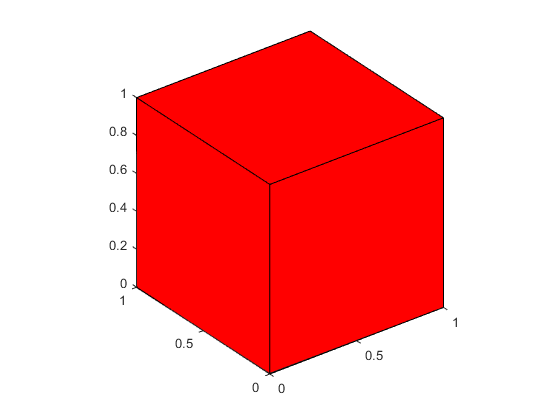
<https://www.povray.org/download/>

# 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:



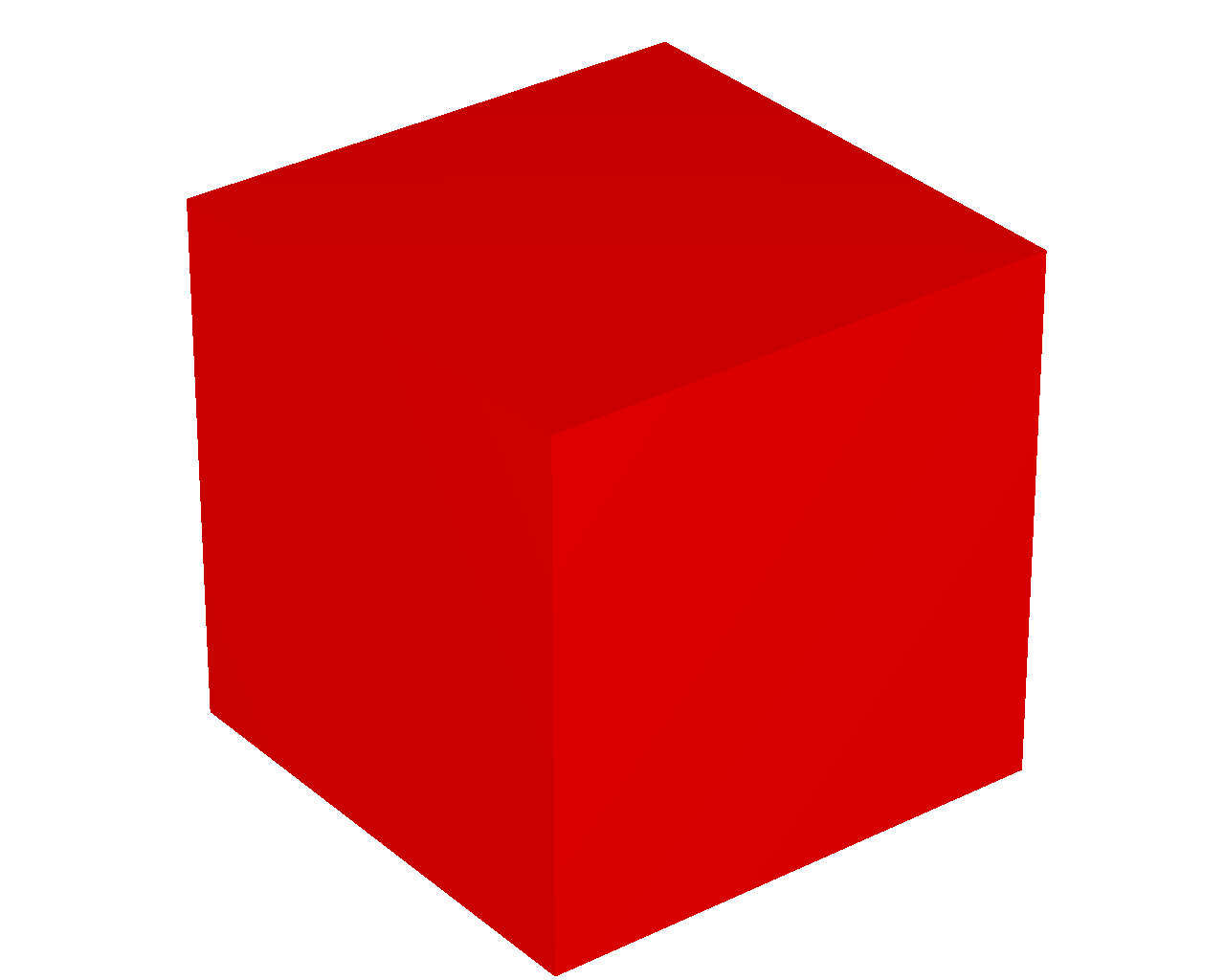
Next, call fig2pov to convert this figure into a Povray script:

|  |
| --- |
| 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 ...

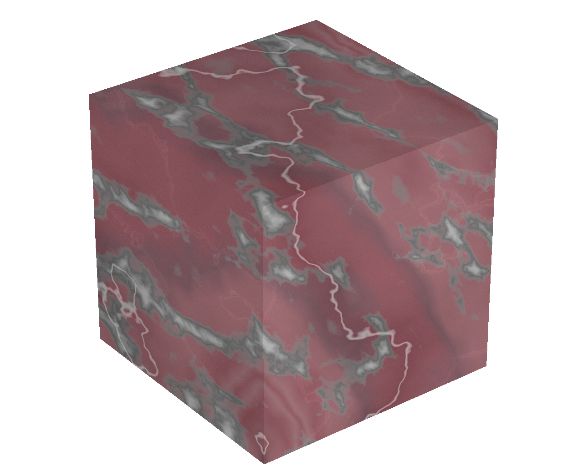
Each graphics object has a property called ‘UserData’, which allows any type of data to be attached to the object. The appearance a graphics object can be changed by adding a structure named 'povray' to the UserData of that object.

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:

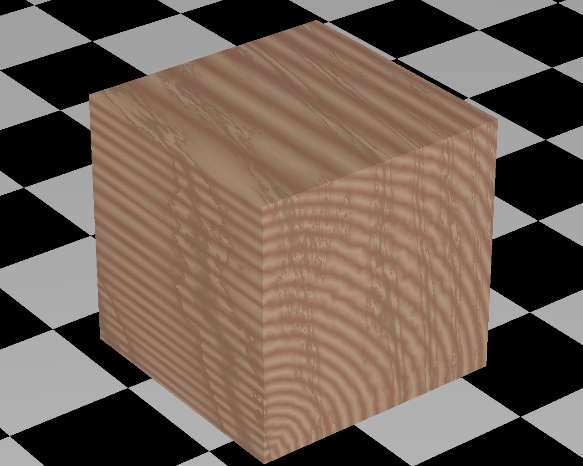


See [here](./Documents) for available textures.

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:



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

The section below provides a full list of all options that can be included in the UserData.povray structure. Different graphics objects may have different options.

# Additional Povray options

Intro text.

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

**Plane**

Adding the field ‘Plane’ to the UserData.povray structure of the axes object draws an infinite plane with the specified orientation. The value of UserData.povray.Plane is an array of four numbers. The first three numbers contain the normal of the Plane. The fourth number is the distance from the origin. For instance [0 0 1 -1] is a plane perpendicular to the Z-axis and crosses the Z-axis at z=-1.

The properties of the plane are further specified by the values of ‘PlaneColor’ and ‘PlaneTexture’

**PlaneColor**

If UserData.povray contains a valid value for the field ‘Plane’, UserData.povray.PlaneColor specifies the color of the Plane. The color can be given as an RGB triplet or as a string.

**PlaneTexture**

If UserData.povray contains a valid value for the field ‘Plane’, UserData.povray.PlaneTexture specifies the texture of the Plane. This value is given as a string containing on of the names of the known Povray textures (see …).

**PlaneTextureScale**

UserData.povray.PlaneTextureScale further specifies the texture of the plane. It only takes effect if both UserData.povray.Plane and UserData.povray.PlaneTexture are contained in UserData.povray. Examples of the use of UserData.povray.PlaneTextureScale are given below for patch objects.

**Define**

IT is possible to define values for use by Povray, for instance to define a custom texture. This can be done by adding the to-be-defined value as a field in UserData.povray of the axes object. The value of the field is the desired definition.

Example:

**Zoom**

fig2pov reproduces the scene as it is contained in the axes. If the overall scale needs to be changed, this can be done by adding the field Zoom to UserData.povray of the axes object. See example below.

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**

UserData.povray.Texture specifies the texture of the patch object. This value is given as a string containing on of the names of the known Povray textures (see …). If the patch has two sides, both Texture and InteriorTexture can be specified. See example below.

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**

UserData.povray of the object contains a valid value for Texture, the appearance of the texture can be further changed by adding the field TextureScale. This field determines the size of the patterns in the scale (e.g. grain of the wood or marbling of the stone). See example below.

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  cube.UserData.povray.Texture = 'T\_Stone18';  cube.UserData.povray.TextureScale = 1; | | |
|  |  |  |
| TextureScale = 1 | TextureScale = 5 | TextureScale = 0.2 |

Default Value: TextureScale = 1.

**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 a 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.

Examples are given later for graphics objects of type surface.

**FaceColor**

FaceColor specifies the color of the patch object. It overwrites the values given in the field FaceColor of the object itself. It can be either an RGB triplet of values or a string.

**drawEdges**

By default, fig2pov creates three-dimensional objects, without edges to trace the different faces of the object. If edges are desired, this can be specified by setting UserData.povray.drawEdges to true. The thickness of the edges is determines by the property ‘LineWidth’ of the patch object.

See example below under EdgeTexture.

**EdgeColor**

If UserData.povray.drawEdges is true, UserData.povray.EdgeColor specifies the color of the edges. It can be either an RGB triplet of values or a string. If no value is given, the color is given by the value of the property EdgeColor of the patch object itself.

See example below under EdgeTexture.

**EdgeTexture**

If UserData.povray.drawEdges is true, UserData.povray.EdgeColor specifies the Texture of the edges. This value is given as a string containing on of the names of the known Povray textures (see …).

See example below.

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  cube.LineWidth = 5;  cube.UserData.povray.drawEdges = true;  cube.UserData.povray.Texture = 'T\_Stone18';  cube.UserData.povray.EdgeTexture = 'Gold\_Texture'; | |
|  |  |
| EdgeTexture = 'Gold\_Texture'  drawEdges = true | drawEdges = true  no EdgeTexture or EdgeColor specified |

**MarkerTexture**

By default, markers are only drawn if the patch properties Marker is different from ‘none’ and the patch properties MarkerFaceColor and MarkerEdgeColor are not both equal to ‘none’. In that case, the markers are drawn as spheres. If UserData.povray.MarkerTexture is specified, the spheres will have the specified texture. See example below.

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  cube.Marker = 'o';  cube.MarkerSize = 12;  cube.MarkerFaceColor = 'r';  cube.UserData.povray.Texture = 'T\_Stone18';  cube.UserData.povray.MarkerTexture = 'Gold\_Texture'; | |
|  |  |
| MarkerTexture = 'Gold\_Texture' | no MarkerTexture specified |

## 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**

Example:

|  |  |
| --- | --- |
| 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 |

Default Value: drawAsCylinder = false

**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 |

Default Value: MeshOn = false

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 |

Default Value: SmoothingOn = false

## 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.

Example:

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

**Texture, TextureScale**

Example:

|  |  |
| --- | --- |
| 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 |

Default Value: no Texture, TextureScale = 1

**SmoothingOn**

Example:

|  |  |
| --- | --- |
| 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 |

Default Value: SmoothingOn = false

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

**ShadowLess**

Example:

|  |  |
| --- | --- |
| [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 |

Default Value: Shadowless = true