How to draw intersecting planes?

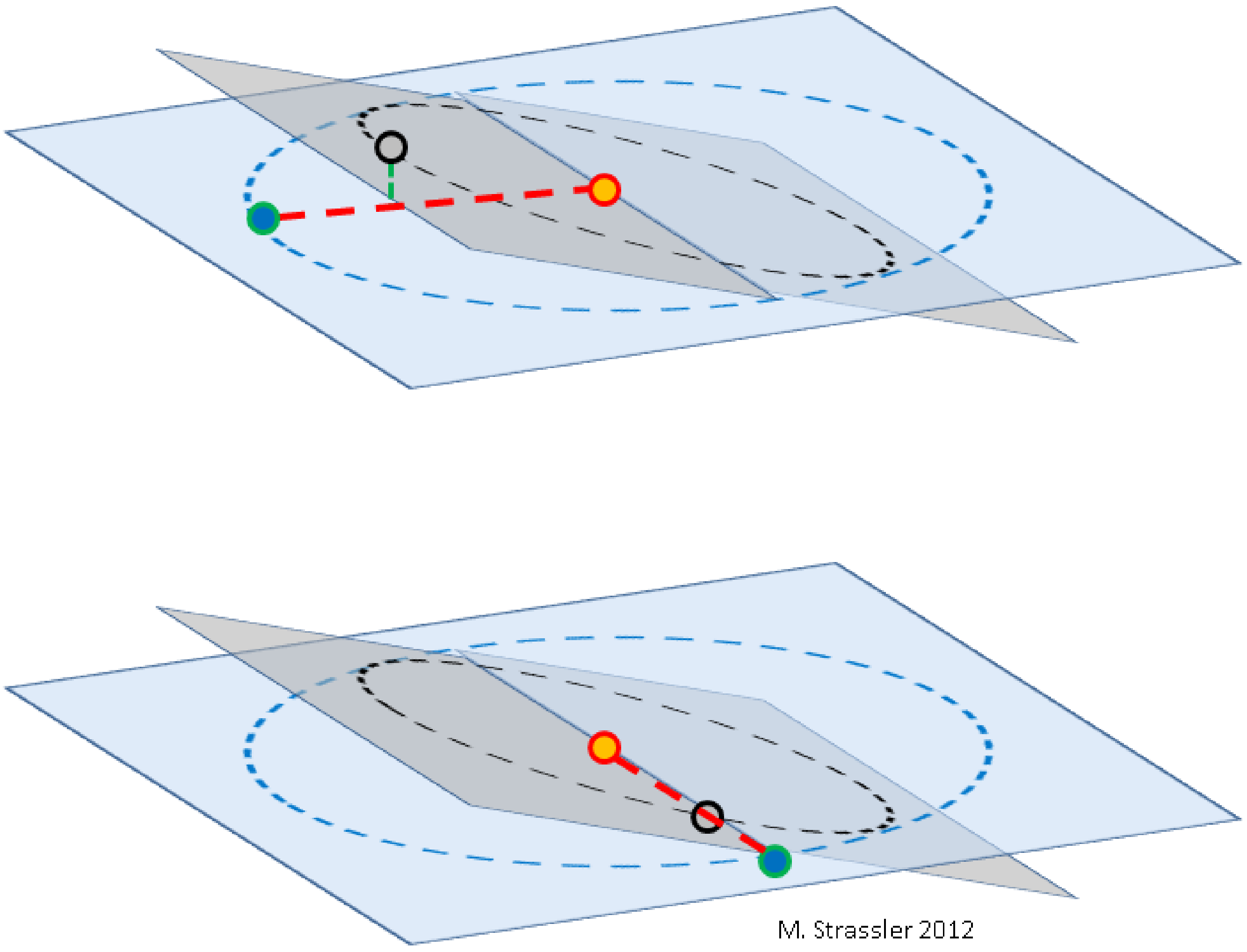
Asked 8 years, 5 months ago Active 5 years, 6 months ago Viewed 12k times

23

16

I want to use matplotlib to draw more or less the figure I attached below, which includes the two intersecting planes with the right amount of transparency indicating their relative orientations, and the circles and vectors in the two planes projected in 2D.

I'm not sure if there is an existing package for doing this, any hints?



python matplotlib Edit tags

Share Edit Follow Close Flag

edited Jan 14 '16 at 19:17
Gabriel
33.3k 60 188 340

asked Feb 12 '13 at 3:26
nye17
11.3k 11 50 62

Does it need to update in real time, just create an image once, fast or can it grind away for a few seconds, entirely automated or is some manual intervention ok? – DarenW Feb 12 '13 at 4:58

1 I'd use NumPy for the projection and pycairo for the graphics. – Ignacio Vazquez-Abrams Feb 12 '13 at 4:58

@DarenW Speed is not a concern here, otherwise I would simply go for 2D plots with b/w points. This is supposed to be a fancy and illustrative diagram. – nye17 Feb 12 '13 at 5:08

@IgnacioVazquez-Abrams that's the last option. I really want something that is smart enough to do the projection for me -- I only need to plot circles and vectors on the plane defined locally and specify the projection angle later on. – nye17 Feb 12 '13 at 5:09

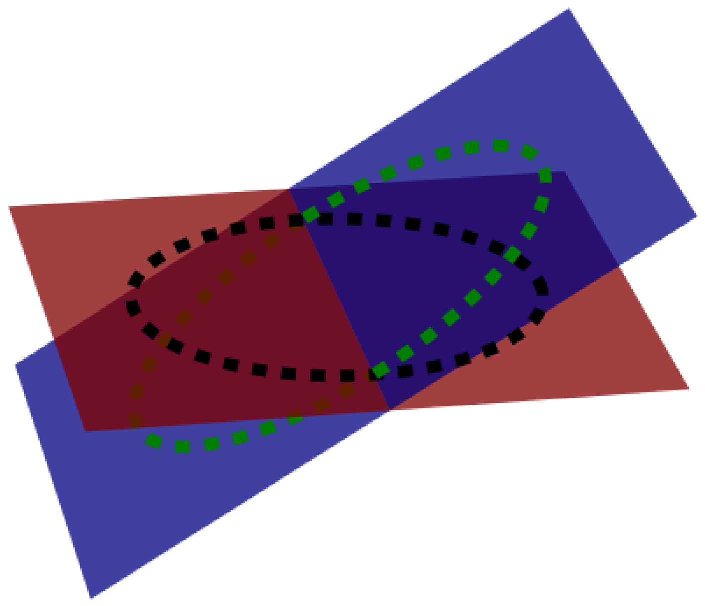
1 Those illustrations have nice perspective on the dashed lines, foreshortened, each dash appearing as if painted on the planes. – DarenW Feb 12 '13 at 5:39

2 Answers

Active Oldest Votes

23

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

dim = 10

X, Y = np.meshgrid([-dim, dim], [-dim, dim])
Z = np.zeros((2, 2))

angle = .5
X2, Y2 = np.meshgrid([-dim, dim], [0, dim])
Z2 = Y2 * angle
X3, Y3 = np.meshgrid([-dim, dim], [-dim, 0])
Z3 = Y3 * angle

r = 7
M = 1000
th = np.linspace(0, 2 * np.pi, M)

x, y, z = r * np.cos(th),  r * np.sin(th), angle * r * np.sin(th)

ax.plot_surface(X2, Y3, Z3, color='blue', alpha=.5, linewidth=0, zorder=-1)

ax.plot(x[y < 0], y[y < 0], z[y < 0], lw=5, linestyle='--', color='green',
        zorder=0)

ax.plot_surface(X, Y, Z, color='red', alpha=.5, linewidth=0, zorder=1)

ax.plot(r * np.sin(th), r * np.cos(th), np.zeros(M), lw=5, linestyle='--',
        color='k', zorder=2)

ax.plot_surface(X2, Y2, Z2, color='blue', alpha=.5, linewidth=0, zorder=3)

ax.plot(x[y > 0], y[y > 0], z[y > 0], lw=5, linestyle='--', color='green',
        zorder=4)

plt.axis('off')
plt.show()
```



caveats:

- I am running a version very close to the current master, so I am not sure what will work in older versions
- The reason for splitting up the plotting is that 'above' and 'below' are determined in a some what arcane way (I am not strictly sure the `zorder` actually does anything), and is really dependent on the order the artists are drawn in. Thus surfaces can not intersect (one will be above the other every where), so you need to plot the sections on either side of the intersection separately. (You can see this in the black line which I didn't split at looks like it in 'on top of' the upper blue plane).
- The 'proper' ordering of the surfaces also seems to be dependent on the view angle.

Share Edit Follow Flag

edited Jun 20 '20 at 9:12
Community ♦
1 1

answered Feb 12 '13 at 5:24
tacaswell
74.6k 16 191 183

Thanks, `zorder` and manual projection seem painful, but good to know that it is absolutely doable in matplotlib. I'll doodle around starting from your code first. – nye17 Feb 12 '13 at 23:53

Note there have been behavior changes in 3.0/3.1 that can affect the ability to render plots like these. I was personally not able to get any combination of zorder to work in my own plots. So if you're having trouble reproducing these results you might try reverting to 2.2.4. See: stackoverflow.com/q/52923540/188046 – Elliott Slaughter Jul 24 '19 at 19:52

You can do this, it just involves some hacky ode, see here. – will Aug 29 '19 at 8:18

5

Matplotlib does have 3d projection capability, but the dashed lines are drawn with constant width in the final 2D image view, not looking as if laid flat on the tilted planes. If the geometry is simple, and the "orbits" circular, it might work, but if you're wanting to draw ellipses seen at an angle, the viewer may desire more visual clues on the whole 3D arrangement.

If I had to make one nice fancy illustration like that, but even nicer and fancier, and it didn't have to be automated, I'd start by creating the graphics - at least the dashed line circles - for each of the planes as a simple flat 2D image using whatever seems handy at the moment - a vector drawing program like Illustrator or Inkscape, or in matplotlib if there is data to be followed.

Then, I'd use POV-Ray or Blender to model the planes at whatever angles, spheres for the round things (planets?). The 2D graphics already generated would become textures to be mapped to the planes. POV-Ray uses a scripting language allowing a record to be kept, modified, and copied for future projets. If it were really one-time and I didn't mind doing it all by hand, Blender is good. Whichever tool I use, the result is an image showing the desired projection of the 3D geometric elements into 2D.

Are the round things, what I'm calling "planets" supposed to be flat circles in the final work, like in the examples? Then I'd draw them with a vector drawing app over the rendered 3D image. But I suspect you'd prefer spheres in 3D.

The samples shown have no lighting or shadows. Shadows would help with clarifying the geometry in 3D, although the first of those two illustrations isn't too bad. The short green line showing the inclined plane's planet over the red line seems clear enough but a shadow would help. The second illustration does look a bit more confusing as to the shape, location an intersections of the various entities. Here, shadows would help more. POV-Ray or Blender will happily create these with little effort. Even more, inter-reflections, known as radiosity, help with seeing 3D relations in 2D images. This advanced effect is easy to do these days, not needing expertise in optics or graphics, just knowing that it exists.

Of course, this advice is no good unless one is already familiar with 3D graphics and tools such s POV-Ray.

For an automated solution, using OpenGL in some quick and dirty program may be best. Shadows may take some work, though.

Share Edit Follow Flag

answered Feb 12 '13 at 5:59
DarenW
15.7k 7 59 98

thanks for your thorough recipe. Although the rendering speed is not a concern here, I do prefer working it out within a single uniformed framework like `matplotlib` --- it seemed like an overkill if I had to learn so many tools for each effect -- I have experience only in python as of now ;-(– nye17 Feb 12 '13 at 23:56