Matplotlib-3D-Animation

April 3, 2025

1 Matplotlib 3D Animation

Arthur Ryman, last updated 2025-04-03

1.1 Introduction

This notebook experiments with the 3D graphics capabilites of matplotlib. This package may provide a quick way to prototype animation scenes which would then be fully rendered using Houdini.

Here is some relevant documentation: * Generate 3D polygons * mpl_toolkits.mplot3d.art3d.Poly3DCollection * Animations using Matplotlib

1.2 Installation

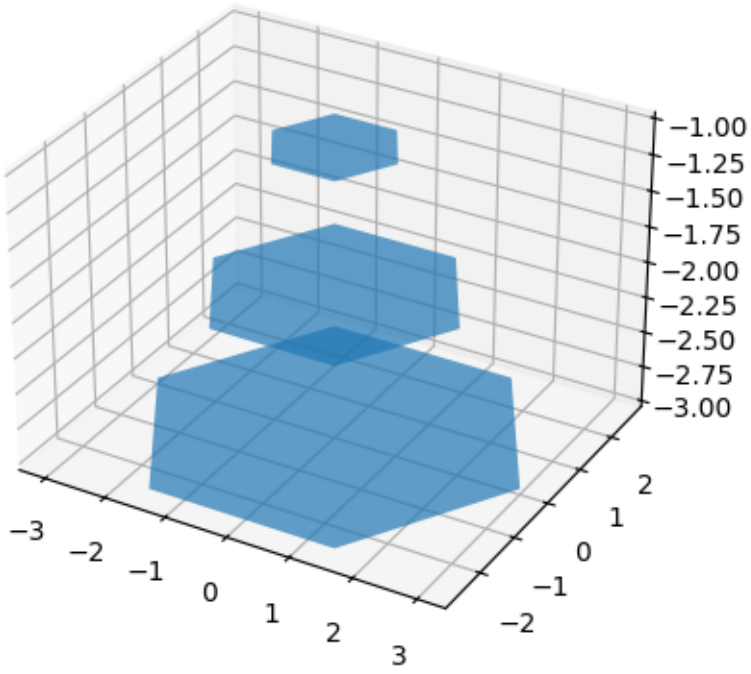
Use `pip` to install `matplotlib` into your Python environment as follows:

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

1.3 Example: Generate 3D polygons

[1]:
```
import matplotlib.pyplot as plt
import numpy as np

from mpl_toolkits.mplot3d.art3d import Poly3DCollection

# Coordinates of a hexagon
angles = np.linspace(0, 2 * np.pi, 6, endpoint=False)
x = np.cos(angles)
y = np.sin(angles)
zs = [-3, -2, -1]

# Close the hexagon by repeating the first vertex
x = np.append(x, x[0])
y = np.append(y, y[0])

verts = []
for z in zs:
```

```
    verts.append(list(zip(x*z, y*z, np.full_like(x, z))))
verts = np.array(verts)

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

poly = Poly3DCollection(verts, alpha=.7)
ax.add_collection3d(poly)
ax.set_aspect('equalxy')

plt.show()
```



[2]:
```
verts
```

[2]:
```
array([[[-3.00000000e+00, -0.00000000e+00, -3.00000000e+00],
        [-1.50000000e+00, -2.59807621e+00, -3.00000000e+00],
        [ 1.50000000e+00, -2.59807621e+00, -3.00000000e+00],
        [ 3.00000000e+00, -3.67394040e-16, -3.00000000e+00],
        [ 1.50000000e+00,  2.59807621e+00, -3.00000000e+00],
        [-1.50000000e+00,  2.59807621e+00, -3.00000000e+00],
        [-3.00000000e+00, -0.00000000e+00, -3.00000000e+00]],

       [[-2.00000000e+00, -0.00000000e+00, -2.00000000e+00],
        [-1.00000000e+00, -1.73205081e+00, -2.00000000e+00],
```

```
 [ 1.00000000e+00, -1.73205081e+00, -2.00000000e+00],
 [ 2.00000000e+00, -2.44929360e-16, -2.00000000e+00],
 [ 1.00000000e+00,  1.73205081e+00, -2.00000000e+00],
 [-1.00000000e+00,  1.73205081e+00, -2.00000000e+00],
 [-2.00000000e+00, -0.00000000e+00, -2.00000000e+00]],

[[-1.00000000e+00, -0.00000000e+00, -1.00000000e+00],
 [-5.00000000e-01, -8.66025404e-01, -1.00000000e+00],
 [ 5.00000000e-01, -8.66025404e-01, -1.00000000e+00],
 [ 1.00000000e+00, -1.22464680e-16, -1.00000000e+00],
 [ 5.00000000e-01,  8.66025404e-01, -1.00000000e+00],
 [-5.00000000e-01,  8.66025404e-01, -1.00000000e+00],
 [-1.00000000e+00, -0.00000000e+00, -1.00000000e+00]]])
```

[ ]:

1.4 Generate a cube

We can generate a cube by drawing its six faces. Each face is a planar surface so we can use `Poly3DCollection` to fill them.

Start by defining the vertices of the cube. Let's use a cube that is centred on the origin and whose sides have length 2. The coordinates of the vertices with therefore be +1 or -1. Centering the cube about the origin makes rotating it easy.

[3]:
```
from itertools import product

vertices = list(product([-1, 1], repeat=3))

vertices
```

[3]:
```
[(-1, -1, -1),
 (-1, -1, 1),
 (-1, 1, -1),
 (-1, 1, 1),
 (1, -1, -1),
 (1, -1, 1),
 (1, 1, -1),
 (1, 1, 1)]
```

[4]:
```
left =[(1,1,1),(1,-1,1),(1,-1,-1),(1,1,-1)]
right=[(-1,-1,1),(-1,1,1),(-1,1,-1),(-1,-1,-1)]

cube = np.array([left,right])
cube
```

[4]:
```
array([[[ 1,  1,  1],
        [ 1, -1,  1],
```

```
        [ 1, -1, -1],
        [ 1,  1, -1]],

       [[-1, -1,  1],
        [-1,  1,  1],
        [-1,  1, -1],
        [-1, -1, -1]]])
```

[5]:
```
ax = plt.figure().add_subplot(projection='3d')

poly = Poly3DCollection(cube, alpha=.7)
ax.add_collection3d(poly)
ax.set_aspect('equalxy')

plt.show()
```