Day 8, Part 1 - intro to ipyvolume

We'll start our journey into the 3RD DIMENSION with the package `ipyvolume`

In [7]:
```
# if you don't get it:
#!pip install ipyvolume
# note: you may need:
#!jupyter nbextension enable --py --sys-prefix ipyvolume
#!jupyter nbextension enable --py --sys-prefix widgetsnbextension

# or you can do:
#!conda install -c conda-forge ipyvolume


import ipyvolume
```

Let's do a quick look at something with some random 3D data:

In [8]:
```
import numpy as np
x, y, z = np.random.random((3, 10000))
ipyvolume.quickscatter(x, y, z, size=1, marker="sphere")
```

Easy peasy! Let's read in our simulation data and plot this!

In [11]:
```
from hermite_library import read_hermite_solution_from_file

# as a test:
t_h, E_h, r_h, v_h = read_hermite_solution_from_file('myPlanetSystem_kep
ler101_solution1.txt')
```

In [12]:
```
# we'll have to reformat a bit for plotting
# right now, just all as one color
x = r_h[:,0,:].ravel()
y = r_h[:,1,:].ravel()
z = r_h[:,2,:].ravel()
ipyvolume.quickscatter(x, y, z,
                       size=1, marker="sphere")
# this plots things as overlapping spheres
# so the orbits look like tubes
```

Let's make things a little more complicated and allow us to take a look at each orbit:

In [13]:

```
ipyvolume.figure()
colors = ['red', 'blue', 'green'] # color of each particle
for i in range(r_h.shape[0]): # loop over particles
    ipyvolume.scatter(r_h[i,0,:],
                      r_h[i,1,:],
                      r_h[i,2,:],
                      color=colors[i],
                     marker='sphere')
ipyvolume.show()
# now all of the orbits are different colors
```

So, this is pretty cool - we can now see how the orbits "precess" during their evolution and we can check out these shapes in 3D.

Note we can also plot more abstract spaces in 3D - like velocity space:

In [14]:

```
v_h
```

Out[14]:

```
array([[[ 1.20141674e-06,  4.46835279e-02,  8.93138660e-02, ...,
         -6.74179312e-01, -6.37677344e-01, -6.00136945e-01],
        [-8.52351194e-01, -8.51372577e-01, -8.48435580e-01, ...,
         -5.87805976e-01, -6.19270706e-01, -6.48672316e-01],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00]],

       [[-1.48139093e-01, -1.65663354e-01, -1.83090972e-01, ...,
         -7.29450017e-01, -7.32358304e-01, -7.34843357e-01],
        [ 7.27438358e-01,  7.23648208e-01,  7.19435966e-01, ...,
          1.28134545e-01,  1.10548127e-01,  9.28952218e-02],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00]],

       [[ 1.20141674e-06, -4.45521515e-06, -1.01058840e-05, ...,
          9.34101445e-05,  8.86965894e-05,  8.38448364e-05],
        [ 1.04715912e-04,  1.04619652e-04,  1.04272661e-04, ...,
          7.52448409e-05,  7.94709091e-05,  8.34297694e-05],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00]]])
```

In [15]:

```
ipyvolume.figure()
colors = ['red', 'blue', 'green'] # now velocity of each particle is dif
ferent color
for i in range(v_h.shape[0]): # loop over number of particles
    ipyvolume.scatter(v_h[i,0,:],
                      v_h[i,1,:],
                      v_h[i,2,:],
                      color=colors[i],
                     marker='sphere')
ipyvolume.show()
```

So this is a little less intiative, but this is how the velocities of our particles change during their orbits.

Ok, we can also show velocity by little vectors:

In [16]:

```
ipyvolume.figure()
colors = ['red', 'blue', 'green'] # each particle's velocity is different color
for i in range(v_h.shape[0]): # loop over particles
    ipyvolume.quiver(r_h[i,0,:], # plot x,y,z positions
                      r_h[i,1,:],
                      r_h[i,2,:],
                     v_h[i,0,:], # also include vx/vy/vz vectors of velocities
                      v_h[i,1,:],
                      v_h[i,2,:],
                      color=colors[i])
ipyvolume.show()
```

So clearly the above is pointless - while it looks cool the arrows are too big and there are too many of them! We can change this by taking "X" number of points. This is like the subsampling we did before to keep our framerates of our animations small:

In [17]:

```
step = 1000 # plot ever "step"th velocity vector
# also, length of arrays in time-axis
N = v_h.shape[2]

ipyvolume.figure()
colors = ['red', 'blue', 'green'] # colors of each particle
for i in range(v_h.shape[0]): # loop every particle
    ipyvolume.quiver(r_h[i,0,0:N:step], # plot subsampled x/y/z
                      r_h[i,1,0:N:step],
                      r_h[i,2,0:N:step],
                     v_h[i,0,0:N:step], # with subsampled vectors vx/vy/vz
                      v_h[i,1,0:N:step],
                      v_h[i,2,0:N:step],
                      color=colors[i],
                     size=2) # also, if things look too crowded, we can also make the arrows themselves smaller
ipyvolume.show()
```

Now we can see a bit more about the motion - that their directions are opposite of eachother for example. And that the central mass only moves slightly and around its center as well.

Animation

Let's now figure out how to make an animation in 3D, and then save it for ourselves! To do this, we'll need to format our data specifically as (time, position):

In [18]:
```
# for example, for particle 0:
r_h[:,0,:].T.shape
```

Out[18]:
```
(5000, 3)
```

In [19]:
```
step = 10 # only do every 10 steps
# also, length of arrays in time
N = v_h.shape[2]

# subsample to make more managable
r = r_h[:,:,0:N:step]
v = v_h[:,:,0:N:step]

r_h.shape, r.shape, r[:,2,:].T.shape
```

Out[19]:
```
((3, 3, 5000), (3, 3, 500), (500, 3))
```

In [20]:
```
# have to format color as well
#colors = np.empty((0,3))
color = [(1,0,0), (0,0,1), (0,1,0)]
```

In [21]:
```
# import little function to do colors for us
from flip_colors import flip_colors

colors = flip_colors(color,r)

colors.shape
```

Out[21]:
```
(500, 3, 3)
```

In [22]:
```
ipyvolume.figure()

s = ipyvolume.scatter(r[:,0,:].T, r[:,1,:].T, r[:,2,:].T,
                      marker='sphere',
                     color=colors)

ani = ipyvolume.animation_control(s, interval=200)

ipyvolume.show()
```

Note that we can only use the `animation_control` function on scatter plots or quiver plots, so we can't add lines or anything here. Perhaps in a future release of `ipyvolume` !

Exercise

Try this with your own datasets!

Bonus: also try with animations of quiver plots

Bonus: is there anything else you want to animate? Should the size of the points change for example? (See ipyvolume docs for examples)

Bonus: do this with the galaxy simulations

Part 2: ipyvolume + ipywidgets

Now let's combine the powers of widgets and ipyvolume to explore our datasets in 3D.

In [23]:
```
import ipywidgets
```

In [24]:
```
step = 100 # only do every 100th timestep
# also, length of arrays
N = v_h.shape[2] # full time

# decimate again
r = r_h[:,:,0:N:step]
v = v_h[:,:,0:N:step]

r[:,0,:].ravel().shape
```

Out[24]:
```
(150,)
```

In [25]:
```
ipyvolume.figure()

x = r[:,0,:].ravel()
y = r[:,1,:].ravel()
z = r[:,2,:].ravel()

s = ipyvolume.scatter(x, y, z,
                      marker='sphere')

#ipyvolume.show()
#colors.shape, r[:,0,:].shape
```

Now let's use widgets to change the size and color of our points:

In [26]:
```
import ipywidgets
size = ipywidgets.FloatSlider(min=0, max=30, step=0.1)
color = ipywidgets.ColorPicker()
```

Now we'll use a function we haven't used before from ipywidgets - something that links our scatter plot features to our widgets:

In [27]:
```
ipywidgets.jslink((s, 'size'), (size, 'value'))
ipywidgets.jslink((s, 'color'), (color, 'value'))
```

```
Link(source=(Scatter(color_selected=array('white', dtype='<U5'), geo='s
phere', line_material=ShaderMaterial(),…
```

Finally, well put all these things in a row: our plot, then our two linked widgets:

In [28]:
```
ipywidgets.VBox([ipyvolume.gcc(), size,  color])
```

Exercise

Repeat this ipywidgets+ipyvolume for your own system.

Bonus: make different sliders for different planets to control size & color for each independently.

Bonus: make a quiver plot

Bonus: what other things can you think to add sliders/pickers for? Hint: check out the docs for `ipyvolume.quiver` and `ipyvolume.scatter` to see what you can change.

Part 3 - embedding

Finally, we might want to embed our creations on the web somewhere. The first step is to make an `html` file from our in-python widgets. Luckily, there is a function for that!

In [29]:
```
myVBox = ipywidgets.VBox([ipyvolume.gcc(), size,  color])
```

In [42]:
```
# if we don't do this, the bqplot will be really tiny in the standalone
 html
ipyvolume.embed.layout = myVBox.children[1].layout
ipyvolume.embed.layout.min_width = "400px"
```

In [43]:
```
# NOTE!!!! offline=True may or may not work... depends
ipyvolume.embed.embed_html("myPage.html", myVBox, offline=False, devmode
=False)
```

In [44]:
```
!open myPage.html
```

Exercise

Generate a page for your own simulation with all the controls you want!

Bonus: though we won't be covering it explicitly, you can actually deploy this to the web to be hosted on github pages. The first thing you need to do is call `embed` a little differently:

In [ ]:
```
ipyvolume.embed.embed_html("myPage.html", myVBox, offline=False, devmode
=False)
```

Now, instead of opening it here, you need to add this file to your github page. Again, we won't cover this in class, but feel free to ask for help after you've looked over the resources provided on today's course webpage under the "deploying to the web" header.

Bonus: add more linkage to your plot by linking to bqplot. See the "Mixing ipyvolume with bqplot" example on the `ipyvolume` docs: https://ipyvolume.readthedocs.io/en/latest/bqplot.html# (https://ipyvolume.readthedocs.io/en/latest/bqplot.html#)

trying animation + widgets