ipyvolume Documentation

Release 0.6.0-alpha.10

Maarten A. Breddels

Nov 01, 2021

Contents

CHAPTER 1

Using pip

Advice: Make sure you use conda or virtualenv. If you are not a root user and want to use the `--user` *argument for pip, you expose the installation to all python environments, which is a bad practice, make sure you know what you are doing.*

```
$ pip install ipyvolume
```

CHAPTER 2

Conda/Anaconda

```
$ conda install -c conda-forge ipyvolume
```

CHAPTER 3

For Jupyter lab users

The Jupyter lab extension is not enabled by default (yet).

```
$ conda install -c conda-forge nodejs  # or some other way to have a recent node
$ jupyter labextension install @jupyter-widgets/jupyterlab-manager
$ jupyter labextension install ipyvolume
$ jupyter labextension install jupyter-threejs
```

CHAPTER 4

Pre-notebook 5.3

If you are still using an old notebook version, ipyvolume and its dependend extension (widgetsnbextension) need to be enabled manually. If unsure, check which extensions are enabled:

```
$ jupyter nbextention list
```

If not enabled, enable them:

```
$ jupyter nbextension enable --py --sys-prefix ipyvolume
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

CHAPTER 5

Pip as user: (but really, do not do this)

You have been warned, do this only if you know what you are doing, this might hunt you in the future, and now is a good time to consider learning virtualenv or conda.

```
$ pip install ipyvolume --user
$ jupyter nbextension enable --py --user ipyvolume
$ jupyter nbextension enable --py --user widgetsnbextension
```

CHAPTER 6

Developer installation

```
$ git clone https://github.com/maartenbreddels/ipyvolume.git
$ cd ipyvolume
$ pip install -e .
$ jupyter nbextension install --py --symlink --sys-prefix ipyvolume
$ jupyter nbextension enable --py --sys-prefix ipyvolume
```

For all cases make sure ipywidgets is enabled if you use Jupyter notebook version < 5.3 (using `--user` instead of `--sys-prefix` if doing a local install):

```
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
$ jupyter nbextension enable --py --sys-prefix pythreejs
$ jupyter nbextension enable --py --sys-prefix ipywebrtc
$ jupyter nbextension enable --py --sys-prefix ipyvolume
```

CHAPTER 7

Developer workflow

7.1 Jupyter notebook (classical)

Note: There is never a need to restart the notebook server, nbextensions are picked up after a page reload.

Start this command:

```
$ (cd js; npm run watch)
```

It will

- Watch for changes in the sourcecode and run the typescript compiler for transpilation of the `src` dir to the `lib` dir.
- Watch the lib dir, and webpack will build (among other things), `ROOT/ipyvolume/static/index.js`.

Refresh the page.

CHAPTER 8

Examples

8.1 Scatter plot

A simple scatter plot, plotting 1000 random points.

```
[1]: import ipyvolume as ipv
     import numpy as np
     N = 1000
     x, y, z = np.random.normal(0, 1, (3, N))
```

```
[2]: fig = ipv.figure()
     scatter = ipv.scatter(x, y, z)
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

screenshot

8.2 Volshow

A simple volume rendering example

8.2.1 Using the pylab API

```
[1]: import numpy as np
     import ipyvolume as ipv
     V = np.zeros((128,128,128)) # our 3d array
     # outer box
     V[30:-30,30:-30,30:-30] = 0.75
```

(continues on next page)

(continued from previous page)

```
V[35:-35,35:-35,35:-35] = 0.0
# inner box
V[50:-50,50:-50,50:-50] = 0.25
V[55:-55,55:-55,55:-55] = 0.0

ipv.figure()
ipv.volshow(V, level=[0.25, 0.75], opacity=0.03, level_width=0.1, data_min=0, data_
↪max=1)
ipv.view(-30, 40)
ipv.show()
```

```
Container(children=[VBox(children=(HBox(children=(Label(value='levels:'),␣
↪FloatSlider(value=0.25, max=1.0, ste...
```

8.3 Visualizating a scan of a male head

Included in ipyvolume, is a visualuzation of a scan of a human head, see the sourcecode for more details.

[2]:
```
import ipyvolume as ipv
fig = ipv.figure()
vol_head = ipv.examples.head(max_shape=128);
vol_head.ray_steps = 400
ipv.view(90, 0)
```

```
Container(children=[HBox(children=(FloatLogSlider(value=1.0, description='opacity',␣
↪max=2.0, min=-2.0), FloatL...
```

[2]:
```
(90, 0, 2.0)
```

screenshot

8.4 Meshes / Surfaces

Meshes (or surfaces) in ipyvolume consist of triangles, and are defined by their coordinate (vertices) and faces/triangles, which refer to three vertices.

[1]:
```
import ipyvolume as ipv
import numpy as np
```

8.4.1 Triangle meshes

Lets first construct a 'solid', a tetrahedron, consisting out of 4 vertices, and 4 faces (triangles) using plot_trisurf

[2]:
```
s = 1/2**0.5
# 4 vertices for the tetrahedron
x = np.array([1.,  -1, 0,  0])
y = np.array([0,   0, 1., -1])
z = np.array([-s, -s, s,  s])
# and 4 surfaces (triangles), where the number refer to the vertex index
triangles = [(0, 1, 2), (0, 1, 3), (0, 2, 3), (1,3,2)]
```

```
[3]: ipv.figure()
     # we draw the tetrahedron
     mesh = ipv.plot_trisurf(x, y, z, triangles=triangles, color='orange')
     # and also mark the vertices
     ipv.scatter(x, y, z, marker='sphere', color='blue')
     ipv.xyzlim(-2, 2)
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.4.2 Surfaces

To draw parametric surfaces, which go from $\mathbb{R}^2 \rightarrow \mathbb{R}^3$, it's convenient to use plot_surface, which takes 2d numpy arrays as arguments, assuming they form a regular grid (meaning you do not need to provide the triangles, since they can be inferred from the shape of the arrays). Note that plot_wireframe has a similar api, as does plot_mesh which can do both the surface and wireframe at the same time.

```
[4]: # f(u, v) -> (u, v, u*v**2)
     a = np.arange(-5, 5)
     U, V = np.meshgrid(a, a)
     X = U
     Y = V
     Z = X*Y**2

     ipv.figure()
     ipv.plot_surface(X, Z, Y, color="orange")
     ipv.plot_wireframe(X, Z, Y, color="red")
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.4.3 Colors

Vertices can take colors as well, as the example below (adapted from matplotlib) shows.

```
[5]: X = np.arange(-5, 5, 0.25*1)
     Y = np.arange(-5, 5, 0.25*1)
     X, Y = np.meshgrid(X, Y)
     R = np.sqrt(X**2 + Y**2)
     Z = np.sin(R)
```

```
[6]: from matplotlib import cm
     colormap = cm.coolwarm
     znorm = Z - Z.min()
     znorm /= znorm.ptp()
     znorm.min(), znorm.max()
     color = colormap(znorm)
```

```
[7]: ipv.figure()
     mesh = ipv.plot_surface(X, Z, Y, color=color[...,:3])
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.4.4 Texture mapping

Texture mapping can be done by providing a PIL image, and UV coordiante (texture coordinates, between 0 and 1). Note that like almost anything in ipyvolume, these u & v coordinates can be animated, as well as the textures.

```
[8]: # import PIL.Image
# image = PIL.Image.open('data/jupyter.png')
# example how put a png as texture
import PIL.Image
import requests
import io

url = 'https://vaex.io/img/logos/spiral-small.png'
r = requests.get(url, stream=True)
f = io.BytesIO(r.content)
image = PIL.Image.open(f)
```

```
[9]: import ipyvolume as ipv
import numpy as np

fig = ipv.figure()
ipv.style.use('dark')
# we create a sequence of 8 u v coordinates so that the texture moves across the␣
↪surface.
u = np.array([X/5 +np.sin(k/8*np.pi)*4. for k in range(8)])
v = np.array([-Y/5*(1-k/7.) + Z*(k/7.) for k in range(8)])
mesh = ipv.plot_mesh(X, Z, Y, u=u, v=v, texture=image, wireframe=False)
ipv.animation_control(mesh, interval=800, sequence_length=8)
ipv.show()
```

```
Container(children=[HBox(children=(Play(value=0, interval=800, max=7),␣
↪FloatSlider(value=0.0, max=7.0, step=1....
```

We now make a small movie / animated gif of 30 frames.

```
[10]: # this doesn't work anymore with modern ipykernel
# frames = 30
# ipv.movie('movie.gif', frames=frames)
```

And play that movie on a square

```
[11]: # so we also need to skip this
# ipv.figure()
# x = np.array([-1.,  1,  1,  -1])
# y = np.array([-1,  -1, 1., 1])
# z = np.array([0., 0, 0., 0])
# u = x / 2 + 0.5
# v = y / 2 + 0.5
# # square
# triangles = [(0, 1, 2), (0, 2, 3)]
# m = ipv.plot_trisurf(x, y, z, triangles=triangles, u=u, v=v, texture=PIL.Image.open(
↪'movie.gif'))
```

(continues on next page)

(continued from previous page)

```
# ipv.animation_control(m, sequence_length=frames)
# ipv.show()
```

screenshot

8.5 Animation

All (or most of) the changes in scatter and quiver plots are (linearly) interpolated. On top top that, scatter plots and quiver plots can take a sequence of arrays (the first dimension), where only one array is visualized. Together this can make smooth animations with coarse timesteps. Lets see an example.

```
[1]: import ipyvolume as ipv
     import numpy as np
```

8.5.1 Basic animation

```
[2]: # only x is a sequence of arrays
     x = np.array([[-1, -0.8], [1, -0.1], [0., 0.5]])
     y = np.array([0.0, 0.0])
     z = np.array([0.0, 0.0])
     ipv.figure()
     s = ipv.scatter(x, y, z, marker='sphere', size=10)
     ipv.xyzlim(-1, 1)
     ipv.animation_control(s) # shows controls for animation controls
     ipv.show()
```

```
Container(children=[HBox(children=(Play(value=0, interval=200, max=2),␣
↪FloatSlider(value=0.0, max=2.0, step=1....
```

You can control which array to visualize, using the `scatter.sequence_index` property. Actually, the `pylab.animate_glyphs` is connecting the `Slider` and `Play` button to that property, but you can also set it from Python.

```
[3]: s.sequence_index = 1
```

8.5.2 Animating color and size

We now demonstrate that you can also animate color and size

```
[4]: # create 2d grids: x, y, and r
     u = np.linspace(-10, 10, 25)
     x, y = np.meshgrid(u, u)
     r = np.sqrt(x**2+y**2)
     print("x,y and z are of shape", x.shape)
     # and turn them into 1d
     x = x.flatten()
     y = y.flatten()
     r = r.flatten()
     print("and flattened of shape", x.shape)
```

```
x,y and z are of shape (25, 25)
and flattened of shape (625,)
```

Now we only animate the z component

```
[5]: # create a sequence of 15 time elements
time = np.linspace(0, np.pi*2, 15)
z = np.array([(np.cos(r + t) * np.exp(-r/5)) for t in time])
print("z is of shape", z.shape)
```

```
z is of shape (15, 625)
```

```
[6]: # draw the scatter plot, and add controls with animate_glyphs
ipv.figure()
s = ipv.scatter(x, z, y, marker="sphere")
ipv.animation_control(s, interval=200)
ipv.ylim(-3,3)
ipv.show()
```

```
Container(children=[HBox(children=(Play(value=0, interval=200, max=14),␣
↪FloatSlider(value=0.0, max=14.0, step=...
```

```
[7]: # Now also include, color, which containts rgb values
color = np.array([[np.cos(r + t), 1-np.abs(z[i]), 0.1+z[i]*0] for i, t in␣
↪enumerate(time)])
size = (z+1)
print("color is of shape", color.shape)
```

```
color is of shape (15, 3, 625)
```

color is of the wrong shape, the last dimension should contain the rgb value, i.e. the shape of should be (15, 2500, 3)

```
[8]: color = np.transpose(color, (0, 2, 1)) # flip the last axes
```

```
[9]: ipv.figure()
s = ipv.scatter(x, z, y, color=color, size=size, marker="sphere")
ipv.animation_control(s, interval=200)
ipv.ylim(-3,3)
ipv.show()
```

```
Container(children=[HBox(children=(Play(value=0, interval=200, max=14),␣
↪FloatSlider(value=0.0, max=14.0, step=...
```

8.5.3 Creating movie files

We now make a movie, with a 2 second duration, where we rotate the camera, and change the size of the scatter points.

```
[10]: # This is commented out, otherwise it would run on readthedocs
# def set_view(figure, framenr, fraction):
#     ipv.view(fraction*360, (fraction - 0.5) * 180, distance=2 + fraction*2)
#     s.size = size * (2+0.5*np.sin(fraction * 6 * np.pi))
# ipv.movie('wave.gif', set_view, fps=20, frames=40)
```

8.5.4 Resulting gif file

8.5.5 Animated quiver

Not only scatter plots can be animated, quiver as well, so the direction vector (vx, vy, vz) can also be animated, as shown in the example below, which is a (subsample of) a simulation of a small galaxy orbiting a host galaxy (not visible).

```
[11]: import ipyvolume.datasets
      stream = ipyvolume.datasets.animated_stream.fetch()
      print("shape of steam data", stream.data.shape) # first dimension contains x, y, z,␣
      ↪vx, vy, vz, then time, then particle
```

```
Downloading https://github.com/maartenbreddels/ipyvolume/raw/master/datasets/stream-
↪animation.npy.bz2 to /home/docs/.ipyvolume/datasets/stream-animation.npy.bz2
```

```
--2021-11-01 12:38:14--  https://github.com/maartenbreddels/ipyvolume/raw/master/
↪datasets/stream-animation.npy.bz2
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/maartenbreddels/ipyvolume/master/datasets/
↪stream-animation.npy.bz2 [following]
--2021-11-01 12:38:14--  https://raw.githubusercontent.com/maartenbreddels/ipyvolume/
↪master/datasets/stream-animation.npy.bz2
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133,␣
↪185.199.111.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:
↪443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7963751 (7.6M) [application/octet-stream]
Saving to: '/home/docs/.ipyvolume/datasets/stream-animation.npy.bz2'

stream-animation.np 100%[===================>]   7.59M  --.-KB/s    in 0.1s

2021-11-01 12:38:14 (67.7 MB/s) - '/home/docs/.ipyvolume/datasets/stream-animation.
↪npy.bz2' saved [7963751/7963751]
```

```
shape of steam data (6, 200, 1250)
```

```
[12]: fig = ipv.figure()
      # instead of doing x=stream.data[0], y=stream.data[1], ... vz=stream.data[5], use␣
      ↪*stream.data
      # limit to 50 timesteps to avoid having a huge notebook
      q = ipv.quiver(*stream.data[:,0:50,:200], color="red", size=7)
      ipv.style.use("dark") # looks better
      ipv.animation_control(q, interval=200)
      ipv.show()
```

```
Container(children=[HBox(children=(Play(value=0, interval=200, max=49),␣
↪FloatSlider(value=0.0, max=49.0, step=...
```

```
[13]: # fig.animation = 0 # set to 0 for no interpolation
```

screenshot

8.6 ipyvolume & bqplot

This example shows how the selection from a ipyvolume quiver plot can be controlled with a bqplot scatter plot and it's selection tools. We first get a small dataset from vaex

```
[1]: import numpy as np
     import vaex
```

```
[2]: ds = vaex.example()
     N = 2000 # for performance reasons we only do a subset
     x, y, z, vx, vy, vz, Lz, E = [ds.columns[k][:N] for k in "x y z vx vy vz Lz E".
     ↪split()]
```

8.6.1 bqplot scatter plot

And create a scatter plot with bqplot

```
[3]: import bqplot.pyplot as plt
```

```
[4]: plt.figure(1, title="E Lz space")
     scatter = plt.scatter(Lz, E,
                     selected_style={'opacity': 0.2, 'size':1, 'stroke': 'red'},
                     unselected_style={'opacity': 0.2, 'size':1, 'stroke': 'blue'},
                     default_size=1,
                    )
     plt.brush_selector()
     plt.show()
```

```
VBox(children=(Figure(axes=[Axis(scale=LinearScale()), Axis(orientation='vertical',␣
↪scale=LinearScale())], fig...
```

8.6.2 Ipyvolume quiver plot

And use ipyvolume to create a quiver plot

```
[5]: import ipyvolume.pylab as ipv
```

```
[6]: ipv.clear()
     quiver = ipv.quiver(x, y, z, vx, vy, vz, size=2, size_selected=5, color_selected="blue
     ↪")
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.6.3 Linking ipyvolume and bqplot

Using jslink, we link the `selected` properties of both widgets, and we display them next to eachother using a VBox.

```
[7]: from ipywidgets import jslink, VBox
```

```
[8]: jslink((scatter, 'selected'), (quiver, 'selected'))
```

```
Link(source=(Scatter(colors=['steelblue'], default_size=1, interactions={'hover':
↪'tooltip'}, scales={'x': Lin...
```

```
[9]: hbox = VBox([ipv.current.container, plt.figure(1)])
# TODO: cannot display the figure twice currently
# hbox
```

8.6.4 Embedding

We embed the two widgets in an html file, creating a standlone plot.

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

```
[11]: ipyvolume.embed.embed_html("bqplot.html", hbox, offline=True, devmode=True)
```

```
Downloading https://unpkg.com/@jupyter-widgets/html-manager@^0.20.0/dist/embed-amd.js:
↪ [==========] Finished
```

```
[12]: %debug
```

```
ERROR:root:No traceback has been produced, nothing to debug.
```

```
[13]: !open bqplot.html
```

```
/bin/sh: 1: open: not found
```

screenshot

8.7 ipyvolume & bokeh

This example shows how the selection from a ipyvolume quiver plot can be controlled with a bokeh scatter plot and it's selection tools.

8.7.1 Ipyvolume quiver plot

The 3d quiver plot is done using ipyvolume

```
[1]: import ipyvolume
import ipyvolume as ipv
import vaex
```

We load some data from vaex, but only use the first 10 000 samples for performance reasons of Bokeh.

```
[2]: ds = vaex.example()
N = 10000
```

```
Downloading https://github.com/vaexio/vaex-datasets/releases/download/v1.0/helmi-
↪dezeeuw-2000-FeH-v2-10percent.hdf5 to /home/docs/.vaex/data/helmi-dezeeuw-2000-FeH-
↪v2-10percent.hdf5
```

```
--2021-11-01 12:38:24--  https://github.com/vaexio/vaex-datasets/releases/download/v1.
↪0/helmi-dezeeuw-2000-FeH-v2-10percent.hdf5
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-releases.githubusercontent.com/242312915/41585000-5723-11ea-
↪9b93-3c6c7ba8ea6c?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
↪Credential=AKIAIWNJYAX4CSVEH53A%2F20211101%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
↪Date=20211101T123824Z&X-Amz-Expires=300&X-Amz-
↪Signature=2b747155e75ac9c366352b3c488d7818ee3982a395e66118e1f7a36cd8e0869c&X-Amz-
↪SignedHeaders=host&actor_id=0&key_id=0&repo_id=242312915&response-content-
↪disposition=attachment%3B%20filename%3Dhelmi-dezeeuw-2000-FeH-v2-10percent.hdf5&
↪response-content-type=application%2Foctet-stream [following]
--2021-11-01 12:38:24--  https://github-releases.githubusercontent.com/242312915/
↪41585000-5723-11ea-9b93-3c6c7ba8ea6c?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
↪Credential=AKIAIWNJYAX4CSVEH53A%2F20211101%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
↪Date=20211101T123824Z&X-Amz-Expires=300&X-Amz-
↪Signature=2b747155e75ac9c366352b3c488d7818ee3982a395e66118e1f7a36cd8e0869c&X-Amz-
↪SignedHeaders=host&actor_id=0&key_id=0&repo_id=242312915&response-content-
↪disposition=attachment%3B%20filename%3Dhelmi-dezeeuw-2000-FeH-v2-10percent.hdf5&
↪response-content-type=application%2Foctet-stream
Resolving github-releases.githubusercontent.com (github-releases.githubusercontent.
↪com)... 185.199.108.154, 185.199.109.154, 185.199.110.154, ...
Connecting to github-releases.githubusercontent.com (github-releases.
↪githubusercontent.com)|185.199.108.154|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13553168 (13M) [application/octet-stream]
Saving to: '/home/docs/.vaex/data/helmi-dezeeuw-2000-FeH-v2-10percent.hdf5'

helmi-dezeeuw-2000- 100%[===================>]  12.92M  --.-KB/s    in 0.1s

2021-11-01 12:38:25 (87.8 MB/s) - '/home/docs/.vaex/data/helmi-dezeeuw-2000-FeH-v2-
↪10percent.hdf5' saved [13553168/13553168]
```

We make a quiver plot using ipyvolume's matplotlib's style api.

[3]:
```
ipv.figure()
quiver = ipv.quiver(ds.data.x[:N],  ds.data.y[:N],  ds.data.z[:N],
                    ds.data.vx[:N], ds.data.vy[:N], ds.data.vz[:N],
                    size=1, size_selected=5, color_selected="grey")
ipv.xyzlim(-30, 30)
ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.7.2 Bokeh scatter part

The 2d scatter plot is done using Bokeh.

[4]:
```
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
from bokeh.models import CustomJS, ColumnDataSource

import ipyvolume.bokeh
output_notebook()
```

```
Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json
```

```
[5]: data_source = ColumnDataSource(data=dict(x=ds.data.Lz[:N], y=ds.data.E[:N]))
```

```
[6]: p = figure(title="E Lz space", tools='lasso_select', width=500, height=500)
     r = p.circle('x', 'y', source=data_source, color="navy", alpha=0.2)
     ipyvolume.bokeh.link_data_source_selection_to_widget(data_source, quiver, 'selected')
     show(p)
```

```
Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json
```

Now try doing a selection and see how the above 3d quiver plot reflects this selection.

```
[7]: # but them next to eachother
     import ipywidgets
     out = ipywidgets.Output()
     with out:
         show(p)
     ipywidgets.HBox([out, ipv.gcc()])
```

```
HBox(children=(Output(), Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_
↪size=[1.0, 1.0, 1.0], camera=...
```

8.7.3 Embedding in html

A bit of a hack, but it is possible to embed the widget and the bokeh part into a single html file (use at own risk).

```
[8]: from bokeh.resources import CDN
     from bokeh.embed import components

     script, div = components((p))
     template_options = dict(extra_script_head=script + CDN.render_js() + CDN.render_css(),
                             body_pre="<h2>Do selections in 2d (bokeh)<h2>" + div + "<h2>
     ↪And see the selection in ipyvolume<h2>")
     ipyvolume.embed.embed_html("tmp/bokeh.html",
                                [ipv.gcc(), ipyvolume.bokeh.wmh], all_states=True,
                                template_options=template_options)
```

```
[9]: # uncomment the next line to open the html file
     # !open tmp/bokeh.html
```

screenshot

8.8 Using scales

Instead of the (default) linear scales, ipyvolume also support bqlot's scales, for instance, the logaritmic scales.

```
[1]: import ipyvolume as ipv
     import bqplot.scales
     import numpy as np
     import ipywidgets as widgets
     N = 500
     x, y, z = np.random.normal(0, 1, (3, N))
     x = 10**x
     r = np.sqrt(np.log10(x)**2 + y**2 + z**2)
```

```
[2]: scales = {
         'x': bqplot.scales.LogScale(min=10**-3, max=10**3),
         'y': bqplot.scales.LinearScale(min=-3, max=3),
         'z': bqplot.scales.LinearScale(min=-3, max=3),
     }
     color_scale = bqplot.scales.ColorScale(min=0, max=3, colors=["#f00", "#0f0", "#00f"])
```

```
[3]: fig = ipv.figure(scales=scales)
     scatter = ipv.scatter(x, y, z, color=r, color_scale=color_scale)
     ipv.view(150, 30, distance=2.5)
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

Note that the x-axis is logarithmically spaced and labeled.

We also use the bqplot color scale, and instead of setting a list of colors, we can also set a famour color scheme:

```
[4]: scatter.color_scale.colors = []
     scatter.color_scale.scheme = 'viridis'
```

Linking a widget to the scale, allows us to easily change its properties

```
[5]: color_max_slider = widgets.FloatSlider(min=1, max=5, description='Color max')
     widgets.jslink((color_scale, 'max'), (color_max_slider, 'value'))
     color_max_slider
```

```
FloatSlider(value=1.0, description='Color max', max=5.0, min=1.0)
```

```
[6]: z_max_slider = widgets.FloatSlider(min=1, max=10, description='Z max')
     widgets.jslink((fig.scales['z'], 'max'), (z_max_slider, 'value'))
     z_max_slider
```

```
FloatSlider(value=1.0, description='Z max', max=10.0, min=1.0)
```

8.8.1 Using the same scales in bqplot

```
[7]: import bqplot.pyplot as plt
     fig2d = plt.figure(layout={'width': '500px'})
     scatter2d = plt.scatter(x=(x), y=y, color=scatter.color, scales={
         "x": fig.scales["x"],
         "y": fig.scales["y"],
         "color": scatter.color_scale
     })
     plt.show()
```

```
VBox(children=(Figure(axes=[Axis(scale=LogScale(max=1000.0, min=0.001)),␣
↪Axis(orientation='vertical', scale=Li...
```

Try zooming/panning in both bqplot and ipyvolume! For ipyvolume, keep the option key pressed, while using scroll or drag.

```
[8]: # Putting the 2 figures next to eachother.
     # TODO: currently not working
     # widgets.VBox([fig, fig2d])
```

screenshot

8.9 Möbius strip

This is a solution for issue #249 and it is the result of contributions from [@deeplok](https://github.com/deeplook) and [@rpanai](https://github.com/rpanai).

```
[1]: from numpy import pi, cos, sin, linspace, meshgrid
     import ipyvolume.pylab as p3

     def möbius(draw=True, show=True, num=40, endpoint=True,
                uv=True, wireframe=False, texture=None):
         # http://paulbourke.net/geometry/toroidal
         u = linspace(0, 2 * pi, num=num, endpoint=endpoint)
         v = linspace(-0.4, 0.4, num=num, endpoint=endpoint)
         u, v = meshgrid(u, v)
         x = cos(u) + v * cos(u / 2) * cos(u)
         y = sin(u) + v * cos(u / 2) * sin(u)
         z = v * sin(u / 2)
         if draw:
             if texture:
                 uv = True
             kwargs = dict(wrapx=not endpoint, wrapy=not endpoint,
                           wireframe=wireframe, texture=texture)
             if uv:
                 kwargs.update(dict(u=u/(2*pi), v=v/(2*pi)))
             mesh = p3.plot_mesh(x, y, z, **kwargs)
             if show:
                 p3.squarelim()
                 p3.show()
             return mesh
         else:
             return x, y, z, u, v

     mesh = möbius()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

screenshot

```
[ ]:
```

8.10 Bar charts

This is 'abusing' the scatter object to create a 3d bar chart

```
[1]: import ipyvolume as ipv
     import numpy as np
```

```
[2]: # set up data similar to animation notebook

     u_scale = 10
     Nx, Ny = 30, 15
     u = np.linspace(-u_scale, u_scale, Nx)
     v = np.linspace(-u_scale, u_scale, Ny)
     x, y = np.meshgrid(u, v, indexing='ij')
     r = np.sqrt(x**2+y**2)
     x = x.flatten()
     y = y.flatten()
     r = r.flatten()

     time = np.linspace(0, np.pi*2, 15)
     z = np.array([(np.cos(r + t) * np.exp(-r/5)) for t in time])
     zz = z
```

```
[ ]:
```

```
[3]: fig = ipv.figure()
     s = ipv.scatter(x, 0, y, aux=zz, marker="sphere")
     dx = u[1] - u[0]
     dy = v[1] - v[0]
     # make the x and z lim half a 'box' larger
     ipv.xlim(-u_scale-dx/2, u_scale+dx/2)
     ipv.zlim(-u_scale-dx/2, u_scale+dx/2)
     ipv.ylim(-1.2, 1.2)
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

We now make boxes, that fit exactly in the volume, by giving them a size of 1, in domain coordinates (so 1 unit as read of by the x-axis etc)

```
[4]: # make the size 1, in domain coordinates (so 1 unit as read of by the x-axis etc)
     s.geo = 'box'
     s.size = 1
     s.size_x_scale = fig.scales['x']
     s.size_y_scale = fig.scales['y']
     s.size_z_scale = fig.scales['z']
```

```
[5]: s.shader_snippets = {'size':
      'size_vector.y = SCALE_SIZE_Y(aux_current); '
     }
```

Using a shader snippet (that runs on the GPU), we set the y size equal to the aux value. However, since the box has size 1 around the origin of (0,0,0), we need to translate it up in the y direction by 0.5.

```
[6]: s.shader_snippets = {'size':
      'size_vector.y = SCALE_SIZE_Y(aux_current) - SCALE_SIZE_Y(0.0) ; '
     }

     s.geo_matrix = [dx, 0, 0, 0,   0, 1, 0, 0,   0, 0, dy, 0,  0.0, 0.5, 0, 1]
```

Since we see the boxes with negative sizes inside out, we made the material double sided

```
[7]: # since we see the boxes with negative sizes inside out, we made the material double␣
     ↪sided
     s.material.side = "DoubleSide"
```

```
[8]: # Now also include, color, which containts rgb values
     color = np.array([[np.cos(r + t), 1-np.abs(z[i]), 0.1+z[i]*0] for i, t in␣
     ↪enumerate(time)])
     color = np.transpose(color, (0, 2, 1)) # flip the last axes
     s.color = color
```

```
[9]: ipv.animation_control(s, interval=200)
```

8.11 Spherical bar charts

```
[10]: # Create spherical coordinates
      u = np.linspace(0, 1, Nx)
      v = np.linspace(0, 1, Ny)
      u, v = np.meshgrid(u, v, indexing='ij')
      phi = u * 2 * np.pi
      theta = v * np.pi
      radius = 1
      xs = radius * np.cos(phi) * np.sin(theta)
      ys = radius * np.sin(phi) * np.sin(theta)
      zs = radius * np.cos(theta)
      xs = xs.flatten()
      ys = ys.flatten()
      zs = zs.flatten()
```

```
[11]: fig = ipv.figure()
      # we use the coordinates as the normals, and thus direction
      s = ipv.scatter(xs, ys, zs, vx=xs, vy=ys, vz=zs, aux=zz, color=color, marker=
      ↪"cylinder_hr")
      ipv.xyzlim(2)
      ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

```
[12]: ipv.animation_control(s, interval=200)
```

```
[13]: import bqplot
      # the aux range is from -1 to 1, but if we put 0 as min, negative values will go␣
      ↪inside
      # the max determines the 'height' of the bars
```

(continues on next page)

(continued from previous page)

```
aux_scale = bqplot.LinearScale(min=0, max=5)
s.aux_scale = aux_scale
```

```
[14]: s.shader_snippets = {'size':
       '''float sc = (SCALE_AUX(aux_current) - SCALE_AUX(0.0)); size_vector.y = sc;
       '''}
      s.material.side = "DoubleSide"
      s.size = 2
      s.geo_matrix = [1, 0, 0, 0,   0, 1, 0, 0,   0, 0, 1, 0,  0.0, 0.5, 0, 1]
```

```
[15]: ipv.style.box_off()
      ipv.style.axes_off()
```

screenshot

8.12 Lightning

By default ipyvolume uses a fake lighting model that works ok for depth perception. To get more realistic lighting effects ipyvolume supports a few light models from pythreejs/threejs.

```
[1]: import ipyvolume as ipv
     import numpy as np
```

```
[2]: def scene():
         f = ipv.figure()
         ipv.xyzlim(-1, 1)
         x = np.array([0.1, 0.5], dtype=np.float32)
         ipv.material_phong()
         s = ipv.scatter(x, x, x, marker="sphere", size=10);
         k = ipv.examples.klein_bottle(show=False)
         ipv.xyzlim(2)
         m = ipv.plot_plane('bottom')
         ipv.show()
         return f
```

8.12.1 Ambient light

For global illumination, which will not cast shadows

```
[3]: scene()
     light = ipv.light_ambient(intensity=0.4);
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.12.2 Hemisphere light

A light source positioned directly above the scene, with color fading from the sky color to the ground color. This light cannot be used to cast shadows.

```
[4]: scene()
     ipv.light_hemisphere(intensity=1);
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.12.3 Directional light

A Directional Light source illuminates all objects equally from a given direction. This light can be used to cast shadows. The light is recommended together with hemisphere.

```
[5]: f = scene()
     ipv.light_ambient(intensity=0.4)
     ipv.light_directional(position=[3, 10, 3]);
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.12.4 Spot light

A Spot Light produces a directed cone of light. The light becomes more intense closer to the spotlight source and to the center of the light cone. This light can be used to cast shadows.

```
[6]: scene()
     ipv.light_hemisphere(intensity=0.2)
     l = ipv.light_spot(position=[0, 3, 2]);
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.12.5 Point light

A Point Light originates from a single point and spreads outward in all directions. This light can be used to cast shadows.

```
[7]: scene()
     ipv.light_hemisphere(intensity=0.2)
     l = ipv.light_point(position=[0., 1.6, 1.0]);
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

8.12.6 Animation example

```
[8]: import ipyvolume as ipv
     import ipyvolume.datasets
     stream = ipyvolume.datasets.animated_stream.fetch()
     print("shape of steam data", stream.data.shape) # first dimension contains x, y, z,␣
     ↪vx, vy, vz, then time, then particle
```

```
shape of steam data (6, 200, 1250)
```

[9]:
```
fig = ipv.figure()
# instead of doing x=stream.data[0], y=stream.data[1], ... vz=stream.data[5], use␣
↪*stream.data
# limit to 50 timesteps to avoid having a huge notebook
ipv.material_physical()
# q = ipv.quiver(*stream.data[:,0:200:10,:2000:10], color="red", size=7)
q = ipv.quiver(*stream.data, color="red", size=7)
ipv.style.use("dark") # looks better
m = ipv.plot_plane('bottom')
m = ipv.plot_plane('back')
m = ipv.plot_plane('left')
ipv
l = ipv.light_directional(position=[20, 50, 20], shadow_camera_orthographic_size=1,␣
↪far=140, near=0.1);
ipv.animation_control(q, interval=200)
ipv.show()
```

```
Container(children=[HBox(children=(Play(value=0, interval=200, max=199),␣
↪FloatSlider(value=0.0, max=199.0, ste...
```

[10]:
```
ipv.light_ambient(intensity=0.4);
```

8.13 Volume rendering and lighting

For volumetric rendering we only support the Phong lighting model. No material needs to be set, only lights need to be setup.

Note that volumetric regions do not cast, not receive shadows.

[11]:
```
import ipyvolume as ipv
```

[12]:
```
# no material and lighting
ipv.figure()
volume = ipv.examples.example_ylm(shape=64, show=False)
s = ipv.scatter(x=32, y=32, z=50, size=30, marker='sphere', color="green");

volume.tf.level1 = 0.24
volume.tf.opacity1 = 0.1
volume.brightness = 2
ipv.show()
```

```
Container(children=[VBox(children=(HBox(children=(Label(value='levels:'),␣
↪FloatSlider(value=0.1, max=1.0, step...
```

[13]:
```
import ipyvolume as ipv
f = ipv.figure(debug=True)
# we only add two lights
ipv.light_hemisphere(intensity=0.8)
ipv.light_directional()

volume = ipv.examples.example_ylm(shape=64, show=False)
ipv.material_phong(specular='white')
s = ipv.scatter(x=32, y=32, z=50, size=30, marker='sphere', color="green");
```

(continues on next page)

(continued from previous page)

```
volume.tf.level1 = 0.24
volume.tf.opacity1 = 0.1
volume.material.specular = 'white'
volume.brightness = 2
ipv.show()
```

```
/home/docs/checkouts/readthedocs.org/user_builds/ipyvolume/envs/latest/lib/python3.7/
↪site-packages/ipykernel_launcher.py:2: DeprecationWarning: debug=True no longer␣
↪needed
```

```
Container(children=[VBox(children=(HBox(children=(Label(value='levels:'),␣
↪FloatSlider(value=0.1, max=1.0, step...
```

screenshot

```
[ ]:
```

8.14 Popups

Ipyvolume has the option to show a popup widgets when hovering above a mark. When hovering, the widget will be shown near the mouse position, and it's `value` attribute will be set to the index of the mark hovered above (e.g. when you have 12 points, value will be between 0 and 11). Also, the description will be set to the description of the scatter object. These two attributes are used in the ipywidget `IntText` and thus can be used as a popop widget:

```
[1]: import ipyvolume as ipv
     import ipywidgets as widgets
     f = ipv.figure()
     scatter = ipv.examples.gaussian(show=False, description="Blob")
     scatter.popup = widgets.IntText()
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

While sufficient, ipyvolume also comes with a custom dedicated Popup widget, build using the ipyvuetify library. This popup will also show a nice icon (see https://materialdesignicons.com/) and the color used.

```
[2]: import ipyvolume as ipv
     import ipywidgets as widgets
     f = ipv.figure()
     scatter = ipv.examples.gaussian(show=False,
                                     description="Blob",
                                     description_color="#CC0000",
                                     icon='mdi-star-four-points')
     scatter.popup = ipv.ui.Popup()
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

Note that while hovering, the scatter attributes `hovered` (a boolean indicates you are hovering above a mark) and `hovered_index`, which mark you are hovering above, are set, and can be linked to other widgets.

```
[3]: widget_hovered = widgets.Valid(description="Hovering", readout="-")
     widget_hovered_index = widgets.Text(description="Hovered on")
     widgets.jsdlink((scatter, 'hovered'), (widget_hovered, 'value'))
     widgets.jsdlink((scatter, 'hovered_index'), (widget_hovered_index, 'value'))
     widgets.HBox([widget_hovered, widget_hovered_index])
```

```
HBox(children=(Valid(value=False, description='Hovering', readout='-'), Text(value='',
↪ description='Hovered on...
```

```
[4]: # workaround for vaex, which has special behaviour on read the docs
     import os
     key = "READTHEDOCS"
     if key in os.environ:
         del os.environ[key]
```

```
[5]: import ipyvolume as ipv
     import vaex.ml
```

```
[6]: df = vaex.ml.datasets.load_iris()
     df
```

```
[6]: #    sepal_length    sepal_width    petal_length    petal_width    class_
     0    5.9             3.0            4.2             1.5            1
     1    6.1             3.0            4.6             1.4            1
     2    6.6             2.9            4.6             1.3            1
     3    6.7             3.3            5.7             2.1            2
     4    5.5             4.2            1.4             0.2            0
     ...  ...             ...            ...             ...            ...
     145  5.2             3.4            1.4             0.2            0
     146  5.1             3.8            1.6             0.2            0
     147  5.8             2.6            4.0             1.2            1
     148  5.7             3.8            1.7             0.3            0
     149  6.2             2.9            4.3             1.3            1
```

```
[7]: import ipywidgets as widgets
     int_widget = widgets.IntText(description="index", value=2)
     int_widget
```

```
IntText(value=2, description='index')
```

```
[8]: import traitlets

     # Custom popup showing a url to wikipedia
     class MyPopup(ipv.ui.Popup):
         # the event handler will fill this in
         template_file = None # disable the loading from file
         url = traitlets.Unicode('').tag(sync=True)
         @traitlets.default("template")
         def _default_template(self):
             return """
         <template>
         <div>
             <div :style="{padding: '4px', 'background-color': color, color: 'white'}">
                 <v-icon color="white">{{icon}}</v-icon>
                 Iris-{{description}}(#<i>{{value}}</i>) <span v-if="extra_html" v-html=
↪"extra_html"></span>
                 <p>
```

(continues on next page)

(continued from previous page)

```
                <a :href="url" target="_black" style="color: white">Visit wikipedia</
↪a>
            </p>

            More information:
            <ul v-if="record" style="margin-top: 0">
                <li v-for="(value, name) in record">{{name}}={{value}}</li>
            </ul>
        </div>
    </div>
    </template>
"""
```

[9]:
```
popup = MyPopup()
classes = ["Setosa", "Versicolour", "Virginica"]
urls = {
    "Setosa": "https://en.wikipedia.org/wiki/Iris_setosa",
    "Versicolour": "https://en.wikipedia.org/wiki/Iris_versicolor",
    "Virginica": "https://en.wikipedia.org/wiki/Iris_virginica"
}

colors = ["red", "green", "blue"]
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

x, y, z = features[:3]
ipv.figure()

for class_index, name in enumerate(classes):
    dfc = df[df.class_==class_index]
    color = colors[class_index]
    s = ipv.scatter(dfc[x].to_numpy(), dfc[y].to_numpy(), dfc[z].to_numpy(),
                    color=color, description=name, marker='sphere')
    s.popup = popup
    def set_extra(index, class_index=class_index, name=name):
        dfc = df[df.class_==class_index]
        records = dfc[features].to_records()
        popup.record = records[index]
        popup.url = urls[name]
    set_extra(0)
    s.observe(set_extra, "hovered")
ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

[10]:
```
# while debugging/developing .vue files in the ipyvolume/vue directory,
# execute this to get hot reloading
# ipv.ui.watch()
```

screencapture

8.15 Slicing

Ipyvolume has no special support for slicing 3d volumes, but has the options to support this. To understand how to do this, we will create a 3d plot, with a plane, which will be controlled using the mouse, and later on texture map this with the same data as the volumetric data.

8.15.1 Simple guassian blob

Lets start with a simple scatter plot.

```
[1]: import ipyvolume as ipv
fig = ipv.figure()
scatter = ipv.examples.gaussian(show=False)
ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

Now, we add a plane at z=0.

```
[2]: plane = ipv.plot_plane("z");
```

By holding the shift key and hovering the mouse at the edges of the bounding box (or activate slice mode in the toolbar, and click), we modify the `slice_z` property. By linking the `slice_z` property to the `z_offset` property of the mesh/plane, we can interactively move the plane. Note that in order to change the `z_offset`, you need to hover the mouse at the sides of the bounding box, which means you need to make sides of the bounding box visible.

```
[3]: import ipywidgets as widgets
widgets.jslink((fig, 'slice_z'), (plane, 'z_offset'));
```

8.15.2 Adding a texture

This plane can be texture mapped with additional information, for instance, a heatmap. We use vaex with maplotlib to create a simple 2d heatmap PIL Image.

```
[4]: ## Uncomment to try
# import vaex
# import matplotlib.pylab as plt
# import PIL.Image

# df = vaex.from_arrays(x=scatter.x, y=scatter.y)

# fig2d = plt.figure()
# ax = fig2d.add_axes([0, 0, 1, 1])
# df.viz.heatmap(df.x, df.y, shape=64, show=False, colorbar=False, tight_layout=False)
# fig2d.axes[0].axis('off');
# plt.draw()
# image = PIL.Image.frombytes('RGB', fig2d.canvas.get_width_height(), fig2d.canvas.
↪tostring_rgb())
# plt.close()
# image
```

On just download an image:

```
[5]: # example how put a png as texture
import PIL.Image
import requests
import io
```

(continues on next page)

(continued from previous page)

```
url = 'https://vaex.io/img/logos/spiral-small.png'
r = requests.get(url, stream=True)
f = io.BytesIO(r.content)
image = PIL.Image.open(f)
```

And assign it to the plane's texture. Note that we should also set its u and v coordinates, so we know where the edges of the texture map should go:

```
[6]: plane.u = [0.0, 1.0, 1.0, 0.0]
     plane.v = [0.0, 0.0, 1.0, 1.0]
     plane.texture = image
```

```
[ ]:
```

8.15.3 Slicing a volume

We can also, texture map a mesh (a plane is a mesh) with a 3d texture, from the volumetric data.

```
[7]: import ipyvolume as ipv
     fig = ipv.figure()
     volume = ipv.examples.head(show=False, description="Patient X")
     ipv.show()
```

```
Downloading https://github.com/maartenbreddels/ipyvolume/raw/master/datasets/male.raw.
↪bz2 to /home/docs/.ipyvolume/datasets/male.raw.bz2
```

```
--2021-11-01 12:38:53--  https://github.com/maartenbreddels/ipyvolume/raw/master/
↪datasets/male.raw.bz2
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/maartenbreddels/ipyvolume/master/datasets/
↪male.raw.bz2 [following]
--2021-11-01 12:38:53--  https://raw.githubusercontent.com/maartenbreddels/ipyvolume/
↪master/datasets/male.raw.bz2
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133,␣
↪185.199.108.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:
↪443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2305035 (2.2M) [application/octet-stream]
Saving to: '/home/docs/.ipyvolume/datasets/male.raw.bz2'

male.raw.bz2        100%[===================>]   2.20M  --.-KB/s    in 0.08s

2021-11-01 12:38:53 (29.3 MB/s) - '/home/docs/.ipyvolume/datasets/male.raw.bz2' saved␣
↪[2305035/2305035]
```

```
Container(children=[HBox(children=(FloatLogSlider(value=1.0, description='opacity',␣
↪max=2.0, min=-2.0), FloatL...
```

We now add 3 planes, and pass our volume so it can be used as a texture map.

```
[8]: slice_x = ipv.plot_plane('x', volume=volume, description="Slice X", description_color=
     ↪"black", icon="mdi-knife")
     slice_y = ipv.plot_plane('y', volume=volume, description="Slice Y", description_color=
     ↪"black", icon="mdi-knife")
     slice_z = ipv.plot_plane('z', volume=volume, description="Slice Z", description_color=
     ↪"black", icon="mdi-knife",
                              visible=False)
```

Again, by connecting the slice coordinates to the offsets of the planes, we can create 3 slicing planes that can be controlled interactively.

```
[9]: import ipywidgets as widgets
     widgets.jslink((fig, 'slice_x'), (slice_x, 'x_offset'))
     widgets.jslink((fig, 'slice_y'), (slice_y, 'y_offset'))
     widgets.jslink((fig, 'slice_z'), (slice_z, 'z_offset'));
```

Note that you can save the output to an html file, and the slicing will still work without a connected kernel.

```
[10]: # uncomment to save
      ipv.save("slice.html", devmode=True)
```

screencapture

```
[ ]:
```

Feel free to contribute new examples:

- Add a notebook to *docs/source/examples*
- Take a screenshot (of screencapture) and put it at *docs/source/examples/screenshot*.
- Make a reference to the screenshot in the notebook, e.g. a markdown cell containing *[screenshot](screenshot/myexample.png)*
- Add an entry in *docs/source/conf.py*.
- Open a pull request at https://github.com/maartenbreddels/ipyvolume

CHAPTER 9

API docs

Note that ipyvolume.pylab and ipyvolume.widgets are imported in the ipyvolume namespace, to you can access ipyvolume.scatter instead of ipyvolume.pylab.scatter.

9.1 Quick list for plotting.

`ipyvolume.pylab.volshow`(data[, lighting, . . .])	Visualize a 3d array using volume rendering.
`ipyvolume.pylab.scatter`(x, y, z[, color, . . .])	Plot many markers/symbols in 3d.
`ipyvolume.pylab.quiver`(x, y, z, u, v, w[, . . .])	Create a quiver plot, which is like a scatter plot but with arrows pointing in the direction given by u, v and w.
`ipyvolume.pylab.plot`(x, y, z[, color, . . .])	Plot a line in 3d.
`ipyvolume.pylab.plot_surface`(x, y, z[, . . .])	Draws a 2d surface in 3d, defined by the 2d ordered arrays x,y,z.
`ipyvolume.pylab.plot_trisurf`(x, y, z[, . . .])	Draw a polygon/triangle mesh defined by a coordinate and triangle indices.
`ipyvolume.pylab.plot_wireframe`(x, y, z[, . . .])	Draws a 2d wireframe in 3d, defines by the 2d ordered arrays x,y,z.
`ipyvolume.pylab.plot_mesh`(x, y, z[, color, . . .])	Draws a 2d wireframe+surface in 3d: generalization of `plot_wireframe` and `plot_surface`.
`ipyvolume.pylab.plot_isosurface`(data[, . . .])	Plot a surface at constant value (like a 2d contour).

9.2 Quick list for controlling the figure.

`ipyvolume.pylab.figure`([key, width, height, . . .])	Create a new figure if no key is given, or return the figure associated with key.
`ipyvolume.pylab.gcf`()	Get current figure, or create a new one.

Continued on next page

Table 2 – continued from previous page

ipyvolume.pylab.gcc()	Return the current container, that is the widget holding the figure and all the control widgets, buttons etc.
ipyvolume.pylab.clear()	Remove current figure (and container).
ipyvolume.pylab.show([extra_widgets])	Display (like in IPython.display.dispay(. . .)) the current figure.
ipyvolume.pylab.view([azimuth, elevation, . . .])	Set camera angles and distance and return the current.
ipyvolume.pylab.xlim(xmin, xmax)	Set limits of x axis.
ipyvolume.pylab.ylim(ymin, ymax)	Set limits of y axis.
ipyvolume.pylab.zlim(zmin, zmax)	Set limits of zaxis.
ipyvolume.pylab.xyzlim(vmin[, vmax])	Set limits or all axis the same, if vmax not given, use [-vmin, vmin].

9.3 Quick list for style and labels.

ipyvolume.pylab.xlabel(label)	Set the labels for the x-axis.
ipyvolume.pylab.ylabel(label)	Set the labels for the y-axis.
ipyvolume.pylab.zlabel(label)	Set the labels for the z-axis.
ipyvolume.pylab.xyzlabel(labelx, labely, labelz)	Set all labels at once.
ipyvolume.pylab.style.use(style)	Set the style of the current figure/visualization.
ipyvolume.pylab.style.set_style_dark()	Short for style.use('dark')
ipyvolume.pylab.style.set_style_light()	Short for style.use('light')
ipyvolume.pylab.style.box_off()	Do not draw the box around the visible volume.
ipyvolume.pylab.style.box_on()	Draw a box around the visible volume.
ipyvolume.pylab.style.axes_off([which])	Do not draw the axes, optionally give axis names, e.g.
ipyvolume.pylab.style.axes_on([which])	Draw the axes, optionally give axis names, e.g.
ipyvolume.pylab.style.background_color(color)	Set the background color.

9.4 Quick list for saving figures.

ipyvolume.pylab.save(filepath[, makedirs, . . .])	Save the current container to a HTML file.
ipyvolume.pylab.savefig(filename[, width, . . .])	Save the figure to an image file.
ipyvolume.pylab.screenshot([width, height, . . .])	Save the figure to a PIL.Image object.

9.5 ipyvolume.pylab

The pylab module of ipvyolume.

ipyvolume.pylab.**scatter**(*x*, *y*, *z*, *color='red'*, *size=2*, *size_selected=2.6*, *color_selected='white'*, *marker='diamond'*, *selection=None*, *grow_limits=True*, *cast_shadow=True*, *receive_shadow=True*, *description=None*, ***kwargs*)

Plot many markers/symbols in 3d. Due to certain shader limitations, should not use with Spot Lights and Point Lights. Does not support shadow mapping.

Parameters

- **x** – numpy array of shape (N,) or (S, N) with x positions. If an (S, N) array, the first dimension will be used for frames in an animation.
- **y** – idem for y
- **z** – idem for z
- **color** – color for each point/vertex/symbol, can be string format, examples for red:'red', '#f00','#ff0000' or'rgb(1,0,0), or rgb array of shape (N, 3 or 4) or (S, N, 3 or 4) Color of the material, essentially a solid color unaffected by other lighting. Default is 'red'
- **size** – float representing the size of the glyph in percentage of the viewport, where 100 is the full size of the viewport
- **size_selected** – like size, but for selected glyphs
- **color_selected** – like color, but for selected glyphs
- **marker** – name of the marker, options are: 'arrow', 'box', 'diamond', 'sphere', 'point_2d', 'square_2d', 'triangle_2d',
- **selection** – numpy array of shape (N,) or (S, N) with indices of x,y,z arrays of the selected markers, which can have a different size and color
- **cast_shadow** – If this object casts a shadown on other options (default) or not. Works only with Directional, Point and Spot lights.
- **receive_shadow** – If this objects receives shadows (default) or not. Works only with Directional, Point and Spot lights.
- **kwargs** –

Returns `Scatter`

`ipyvolume.pylab.`**`quiver`**(*x*, *y*, *z*, *u*, *v*, *w*, *size=20*, *size_selected=26.0*, *color='red'*, *color_selected='white'*, *marker='arrow'*, *cast_shadow=True*, *receive_shadow=True*, ***kwargs*)

Create a quiver plot, which is like a scatter plot but with arrows pointing in the direction given by u, v and w.

Parameters

- **x** – numpy array of shape (N,) or (S, N) with x positions. If an (S, N) array, the first dimension will be used for frames in an animation.
- **y** – idem for y
- **z** – idem for z
- **u** – numpy array of shape (N,) or (S, N) indicating the x component of a vector. If an (S, N) array, the first dimension will be used for frames in an animation.
- **v** – idem for y
- **w** – idem for z
- **size** – float representing the size of the glyph in percentage of the viewport, where 100 is the full size of the viewport
- **size_selected** – like size, but for selected glyphs

- **`color`** – color for each point/vertex/symbol, can be string format, examples for red:'red', '#f00','#ff0000' or'rgb(1,0,0), or rgb array of shape (N, 3 or 4) or (S, N, 3 or 4)
- **`color_selected`** – like color, but for selected glyphs
- **`marker`** – (currently only 'arrow' would make sense)
- **`cast_shadow`** – If this object casts a shadown on other options (default) or not. Works only with Directional, Point and Spot lights.
- **`receive_shadow`** – If this objects receives shadows (default) or not. Works only with Directional, Point and Spot lights.
- **`kwargs`** – extra arguments passed on to the Scatter constructor

Returns `Scatter`

`ipyvolume.pylab.`**`plot`**(*x*, *y*, *z*, *color='red'*, *cast_shadow=True*, *receive_shadow=True*, ***kwargs*)

Plot a line in 3d.

Parameters

- **`x`** – numpy array of shape (N,) or (S, N) with x positions. If an (S, N) array, the first dimension will be used for frames in an animation.
- **`y`** – idem for y
- **`z`** – idem for z
- **`color`** – color for each point/vertex/symbol, can be string format, examples for red:'red', '#f00','#ff0000' or'rgb(1,0,0), or rgb array of shape (N, 3 or 4) or (S, N, 3 or 4)
- **`cast_shadow`** – If this object casts a shadown on other options (default) or not. Works only with Directional, Point and Spot lights.
- **`receive_shadow`** – If this objects receives shadows (default) or not. Works only with Directional, Point and Spot lights.
- **`kwargs`** – extra arguments passed to the Scatter constructor

Returns `Scatter`

`ipyvolume.pylab.`**`volshow`**(*data, lighting=False, data_min=None, data_max=None, max_shape=256, tf=None, stereo=False, ambient_coefficient=0.5, diffuse_coefficient=0.8, specular_coefficient=0.5, specular_exponent=5, downscale=1, level=[0.1, 0.5, 0.9], opacity=[0.01, 0.05, 0.1], level_width=0.1, controls=True, max_opacity=0.2, memorder='C', extent=None, description=None*)

Visualize a 3d array using volume rendering.

Currently only 1 volume can be rendered.

Parameters

- **`data`** – 3d numpy array
- **`origin`** – origin of the volume data, this is to match meshes which have a different origin
- **`domain_size`** – domain size is the size of the volume
- **`lighting`** (`bool`) – use lighting or not, if set to false, lighting parameters will be overriden
- **`data_min`** (`float`) – minimum value to consider for data, if None, computed using np.nanmin
- **`data_max`** (`float`) – maximum value to consider for data, if None, computed using np.nanmax

- **tf** – transfer function (or a default one)
- **stereo** (*bool*) – stereo view for virtual reality (cardboard and similar VR head mount)
- **ambient_coefficient** – lighting parameter
- **diffuse_coefficient** – lighting parameter
- **specular_coefficient** – lighting parameter
- **specular_exponent** – lighting parameter
- **downscale** (*float*) – downscale the rendering for better performance, for instance when set to 2, a 512x512 canvas will show a 256x256 rendering upscaled, but it will render twice as fast.
- **level** – level(s) for the where the opacity in the volume peaks, maximum sequence of length 3
- **opacity** – opacity(ies) for each level, scalar or sequence of max length 3
- **level_width** – width of the (gaussian) bumps where the opacity peaks, scalar or sequence of max length 3
- **controls** (*bool*) – add controls for lighting and transfer function or not
- **max_opacity** (*float*) – maximum opacity for transfer function controls
- **extent** – list of [[xmin, xmax], [ymin, ymax], [zmin, zmax]] values that define the bounds of the volume, otherwise the viewport is used
- **description** – Used in the legend and in popup to identify the object

Parap int max_shape maximum shape for the 3d cube, if larger, the data is reduced by skipping/slicing (data[::N]), set to None to disable.

Returns

`ipyvolume.pylab.`**`plot_surface`**(*x*, *y*, *z*, *color='red'*, *wrapx=False*, *wrapy=False*, *cast_shadow=True*, *receive_shadow=True*)

Draws a 2d surface in 3d, defined by the 2d ordered arrays x,y,z.

Parameters

- **x** – numpy array of shape (N,M) or (S, N, M) with x positions. If an (S, N, M) array, the first dimension will be used for frames in an animation.
- **y** – idem for y
- **z** – idem for z
- **color** – color for each point/vertex string format, examples for red:'red', '#f00', '#ff0000' or 'rgb(1,0,0), or rgb
- **wrapx** (*bool*) – when True, the x direction is assumed to wrap, and polygons are drawn between the end end begin points
- **wrapy** (*bool*) – simular for the y coordinate
- **cast_shadow** – If this object casts a shadown on other options (default) or not. Works only with Directional, Point and Spot lights.
- **receive_shadow** – If this objects receives shadows (default) or not. Works only with Directional, Point and Spot lights.

Returns *Mesh*

`ipyvolume.pylab.`**`plot_trisurf`**(*x*, *y*, *z*, *triangles=None*, *lines=None*, *color='red'*, *u=None*, *v=None*, *texture=None*, *cast_shadow=True*, *receive_shadow=True*, *description=None*, ***kwargs*)

Draw a polygon/triangle mesh defined by a coordinate and triangle indices.

The following example plots a rectangle in the z==2 plane, consisting of 2 triangles:

```
>>> plot_trisurf([0, 0, 3., 3.], [0, 4., 0, 4.], 2,
       triangles=[[0, 2, 3], [0, 3, 1]])
```

Note that the z value is constant, and thus not a list/array. For guidance, the triangles refer to the vertices in this manner:

```
^ ydir
|
2 3
0 1  ---> x dir
```

Note that if you want per face/triangle colors, you need to duplicate each vertex.

Parameters

- **`x`** – numpy array of shape (N,) or (S, N) with x positions. If an (S, N) array, the first dimension will be used for frames in an animation.
- **`y`** – idem for y
- **`z`** – idem for z
- **`triangles`** – numpy array with indices referring to the vertices, defining the triangles, with shape (M, 3)
- **`lines`** – numpy array with indices referring to the vertices, defining the lines, with shape (K, 2)
- **`color`** – color for each point/vertex/symbol, can be string format, examples for red:'red', '#f00','#ff0000' or'rgb(1,0,0), or rgb array of shape (N, 3 or 4) or (S, N, 3 or 4) Color of the material, essentially a solid color unaffected by other lighting. Default is 'red'
- **`u`** – numpy array of shape (N,) or (S, N) indicating the u (x) coordinate for the texture. If an (S, N) array, the first dimension will be used for frames in an animation.
- **`v`** – numpy array of shape (N,) or (S, N) indicating the v (y) coordinate for the texture. If an (S, N) array, the first dimension will be used for frames in an animation.
- **`texture`** – PIL.Image object or ipywebrtc.MediaStream (can be a seqence)
- **`cast_shadow`** – If this object casts a shadown on other options (default) or not. Works only with Directional, Point and Spot lights.
- **`receive_shadow`** – If this objects receives shadows (default) or not. Works only with Directional, Point and Spot lights.
- **`description`** – Used in the legend and in popup to identify the object

Returns *`Mesh`*

`ipyvolume.pylab.`**`plot_wireframe`**(*x*, *y*, *z*, *color='red'*, *wrapx=False*, *wrapy=False*, *cast_shadow=True*, *receive_shadow=True*)

Draws a 2d wireframe in 3d, defines by the 2d ordered arrays x,y,z.

See also *`ipyvolume.pylab.plot_mesh`*

Parameters

- **x** – numpy array of shape (N,M) or (S, N, M) with x positions. If an (S, N, M) array, the first dimension will be used for frames in an animation.
- **y** – idem for y
- **z** – idem for z
- **color** – color for each point/vertex string format, examples for red:'red', '#f00', '#ff0000' or 'rgb(1,0,0), or rgb
- **wrapx** (*bool*) – when True, the x direction is assumed to wrap, and polygons are drawn between the begin and end points
- **wrapy** (*bool*) – idem for y
- **cast_shadow** – If this object casts a shadown on other options (default) or not. Works only with Directional, Point and Spot lights.
- **receive_shadow** – If this objects receives shadows (default) or not. Works only with Directional, Point and Spot lights.

Returns *Mesh*

`ipyvolume.pylab.`**`plot_mesh`**(*x*, *y*, *z*, *color='red'*, *wireframe=True*, *surface=True*, *wrapx=False*, *wrapy=False*, *u=None*, *v=None*, *texture=None*, *cast_shadow=True*, *receive_shadow=True*, *description=None*)

Draws a 2d wireframe+surface in 3d: generalization of `plot_wireframe` and `plot_surface`.

Parameters

- **x** – numpy array of shape (N,M) or (S, N, M) with x positions. If an (S, N, M) array, the first dimension will be used for frames in an animation.
- **y** – idem for y
- **z** – idem for z
- **color** – color for each point/vertex string format, examples for red:'red', '#f00', '#ff0000' or 'rgb(1,0,0), or rgb
- **wireframe** (*bool*) – draw lines between the vertices
- **surface** (*bool*) – draw faces/triangles between the vertices
- **wrapx** (*bool*) – when True, the x direction is assumed to wrap, and polygons are drawn between the begin and end points
- **wrapy** (*booool*) – idem for y
- **u** – numpy array of shape (N,) or (S, N) indicating the u (x) coordinate for the texture. If an (S, N) array, the first dimension will be used for frames in an animation.
- **v** – numpy array of shape (N,) or (S, N) indicating the v (y) coordinate for the texture. If an (S, N) array, the first dimension will be used for frames in an animation.
- **texture** – PIL.Image object or ipywebrtc.MediaStream (can be a seqence)
- **cast_shadow** – If this object casts a shadown on other options (default) or not. Works only with Directional, Point and Spot lights.
- **receive_shadow** – If this objects receives shadows (default) or not. Works only with Directional, Point and Spot lights.
- **description** – Used in the legend and in popup to identify the object

Returns *Mesh*

`ipyvolume.pylab.`**`plot_isosurface`**(*data*, *level=None*, *color='red'*, *wireframe=True*, *surface=True*, *controls=True*, *extent=None*, *description=None*)

Plot a surface at constant value (like a 2d contour).

Parameters

- **`data`** – 3d numpy array
- **`level`** (*`float`*) – value where the surface should lie
- **`color`** – color of the surface, although it can be an array, the length is difficult to predict beforehand, if per vertex color are needed, it is better to set them on the returned mesh afterwards.
- **`wireframe`** (*`bool`*) – draw lines between the vertices
- **`surface`** (*`bool`*) – draw faces/triangles between the vertices
- **`controls`** (*`bool`*) – add controls to change the isosurface
- **`extent`** – list of [[xmin, xmax], [ymin, ymax], [zmin, zmax]] values that define the bounding box of the mesh, otherwise the viewport is used
- **`description`** – Used in the legend and in popup to identify the object

Returns *`Mesh`*

`ipyvolume.pylab.`**`xlim`**(*xmin*, *xmax*)

Set limits of x axis.

`ipyvolume.pylab.`**`ylim`**(*ymin*, *ymax*)

Set limits of y axis.

`ipyvolume.pylab.`**`zlim`**(*zmin*, *zmax*)

Set limits of zaxis.

`ipyvolume.pylab.`**`xyzlim`**(*vmin*, *vmax=None*)

Set limits or all axis the same, if vmax not given, use [-vmin, vmin].

`ipyvolume.pylab.`**`xlabel`**(*label*)

Set the labels for the x-axis.

`ipyvolume.pylab.`**`ylabel`**(*label*)

Set the labels for the y-axis.

`ipyvolume.pylab.`**`zlabel`**(*label*)

Set the labels for the z-axis.

`ipyvolume.pylab.`**`xyzlabel`**(*labelx*, *labely*, *labelz*)

Set all labels at once.

`ipyvolume.pylab.`**`view`**(*azimuth=None*, *elevation=None*, *distance=None*)

Set camera angles and distance and return the current.

Parameters

- **`azimuth`** (*`float`*) – rotation around the axis pointing up in degrees
- **`elevation`** (*`float`*) – rotation where +90 means 'up', -90 means 'down', in degrees
- **`distance`** (*`float`*) – radial distance from the center to the camera.

`ipyvolume.pylab.`**`figure`**(*key=None*, *width=400*, *height=500*, *lighting=True*, *controls=True*, *controls_vr=False*, *controls_light=False*, *debug=False*, ***kwargs*)

Create a new figure if no key is given, or return the figure associated with key.

Parameters

- **key** – Python object that identifies this figure
- **width** (*int*) – pixel width of WebGL canvas
- **height** (*int*) –
- **lighting** (*bool*) – use lighting or not
- **controls** (*bool*) – show controls or not
- **controls_vr** (*bool*) – show controls for VR or not
- **debug** (*bool*) – show debug buttons or not

Returns *Figure*

`ipyvolume.pylab.`**`gcf`**`()`

Get current figure, or create a new one.

Returns *Figure*

`ipyvolume.pylab.`**`gcc`**`()`

Return the current container, that is the widget holding the figure and all the control widgets, buttons etc.

`ipyvolume.pylab.`**`clear`**`()`

Remove current figure (and container).

`ipyvolume.pylab.`**`show`**`(`*extra_widgets=[]*`)`

Display (like in IPython.display.dispay(. . .)) the current figure.

`ipyvolume.pylab.`**`save`**`(`*filepath, makedirs=True, title='IPyVolume Widget', all_states=False, offline=False, scripts_path='js', drop_defaults=False, template_options=(('extra_script_head', ''), ('body_pre', ''), ('body_post', '')), devmode=False, offline_cors=False*`)`

Save the current container to a HTML file.

By default the HTML file is not standalone and requires an internet connection to fetch a few javascript libraries. Use offline=True to download these and make the HTML file work without an internet connection.

Parameters

- **filepath** (*str*) – The file to write the HTML output to.
- **makedirs** (*bool*) – whether to make directories in the filename path, if they do not already exist
- **title** (*str*) – title for the html page
- **all_states** (*bool*) – if True, the state of all widgets know to the widget manager is included, else only those in widgets
- **offline** (*bool*) – if True, use local urls for required js/css packages and download all js/css required packages (if not already available), such that the html can be viewed with no internet connection
- **scripts_path** (*str*) – the folder to save required js/css packages to (relative to the filepath)
- **drop_defaults** (*bool*) – Whether to drop default values from the widget states
- **template_options** – list or dict of additional template options
- **devmode** (*bool*) – if True, attempt to get index.js from local js/dist folder
- **offline_cors** (*bool*) – if True, sets crossorigin attribute of script tags to anonymous

`ipyvolume.pylab.`**`savefig`**(*filename*, *width=None*, *height=None*, *fig=None*, *timeout_seconds=10*, *output_widget=None*, *headless=False*, *devmode=False*)

Save the figure to an image file.

Parameters

- **filename** (*str*) – must have extension .png, .jpeg or .svg
- **width** (*int*) – the width of the image in pixels
- **height** (*int*) – the height of the image in pixels
- **fig** (ipyvolume.widgets.Figure *or None*) – if None use the current figure
- **timeout_seconds** (*float*) – maximum time to wait for image data to return
- **output_widget** (*ipywidgets.Output*) – a widget to use as a context manager for capturing the data
- **headless** (*bool*) – if True, use headless chrome to save figure
- **devmode** (*bool*) – if True, attempt to get index.js from local js/dist folder

`ipyvolume.pylab.`**`screenshot`**(*width=None*, *height=None*, *format='png'*, *fig=None*, *timeout_seconds=10*, *output_widget=None*, *headless=False*, *devmode=False*)

Save the figure to a PIL.Image object.

Parameters

- **width** (*int*) – the width of the image in pixels
- **height** (*int*) – the height of the image in pixels
- **format** – format of output data (png, jpeg or svg)
- **fig** (ipyvolume.widgets.Figure *or None*) – if None use the current figure
- **timeout_seconds** (*int*) – maximum time to wait for image data to return
- **output_widget** (*ipywidgets.Output*) – a widget to use as a context manager for capturing the data
- **headless** (*bool*) – if True, use headless chrome to take screenshot
- **devmode** (*bool*) – if True, attempt to get index.js from local js/dist folder

Returns PIL.Image

`ipyvolume.pylab.`**`selector_default`**(*output_widget=None*)

Capture selection events from the current figure, and apply the selections to Scatter objects.

Example:

```
>>> import ipyvolume as ipv
>>> ipv.figure()
>>> ipv.examples.gaussian()
>>> ipv.selector_default()
>>> ipv.show()
```

Now hold the control key to do selections, type

- 'C' for circle
- 'R' for rectangle
- 'L' for lasso

- '=' for replace mode
- '&' for logically and mode
- '|' for logically or mode
- '-' for subtract mode

`ipyvolume.pylab.`**`movie`**(*f='movie.mp4', function=<function _change_azimuth_angle>, fps=30, frames=30, endpoint=False, cmd_template_ffmpeg='ffmpeg -y -r {fps} -i {tempdir}/frame-%5d.png -vcodec h264 -pix_fmt yuv420p {filename}', cmd_template_gif='convert -delay {delay} {loop} {tempdir}/frame-*.png {filename}', gif_loop=0*)

Create a movie out of many frames in e.g. mp4 or gif format.

If the filename ends in *.gif*, *convert* is used to convert all frames to an animated gif using the *cmd_template_gif* template. Otherwise *ffmpeg is assumed to know the file format*.

Example:

```
>>> def set_angles(fig, i, fraction):
>>>     fig.angley = fraction*np.pi*2
>>> # 4 second movie, that rotates around the y axis
>>> p3.movie('test2.gif', set_angles, fps=20, frames=20*4,
        endpoint=False)
```

Note that in the example above we use *endpoint=False* to avoid to first and last frame to be the same

Parameters

- **`f`** (*`str`*) – filename out output movie (e.g. 'movie.mp4' or 'movie.gif')
- **`function`** – function called before each frame with arguments (figure, framenr, fraction)
- **`fps`** – frames per seconds
- **`frames`** (*`int`*) – total number of frames
- **`endpoint`** (*`bool`*) – if fraction goes from [0, 1] (inclusive) or [0, 1) (endpoint=False is useful for loops/rotatations)
- **`cmd_template_ffmpeg`** (*`str`*) – template command when running ffmpeg (non-gif ending filenames)
- **`cmd_template_gif`** (*`str`*) – template command when running imagemagick's convert (if filename ends in .gif)
- **`gif_loop`** – None for no loop, otherwise the framenumber to go to after the last frame

Returns the temp dir where the frames are stored

`ipyvolume.pylab.`**`animation_control`**(*object, sequence_length=None, add=True, interval=200*)

Animate scatter, quiver or mesh by adding a slider and play button.

Parameters

- **`object`** – *`Scatter`* or *`Mesh`* object (having an sequence_index property), or a list of these to control multiple.
- **`sequence_length`** – If sequence_length is None we try try our best to figure out, in case we do it badly, you can tell us what it should be. Should be equal to the S in the shape of the numpy arrays as for instance documented in *`scatter`* or *`plot_mesh`*.
- **`add`** – if True, add the widgets to the container, else return a HBox with the slider and play button. Useful when you want to customise the layout of the widgets yourself.

- **interval** – interval in msec between each frame

Returns If add is False, if returns the ipywidgets.HBox object containing the controls

ipyvolume.pylab.**transfer_function**(*level=[0.1, 0.5, 0.9], opacity=[0.01, 0.05, 0.1], level_width=0.1, controls=True, max_opacity=0.2*)

Create a transfer function, see volshow.

class ipyvolume.pylab.**style**

Bases: object

Static class that mimics a matplotlib module.

Example:

```
>>> import ipyvolume as ipv
>>> ipv.style.use('light'])
>>> ipv.style.use('seaborn-darkgrid'])
>>> ipv.style.use(['seaborn-darkgrid', {'axes.x.color':'orange'}])
```

Possible style values:

- figure.facecolor: background color
- axes.color: color of the box around the volume/viewport
- xaxis.color: color of xaxis
- yaxis.color: color of xaxis
- zaxis.color: color of xaxis

static axes_off(*which=None*)

Do not draw the axes, optionally give axis names, e.g. 'xy'.

static axes_on(*which=None*)

Draw the axes, optionally give axis names, e.g. 'xy'.

static background_color(*color*)

Set the background color.

static box_off()

Do not draw the box around the visible volume.

static box_on()

Draw a box around the visible volume.

static set_style_dark()

Short for style.use('dark')

static set_style_demo()

Short for style.use('demo')

static set_style_light()

Short for style.use('light')

static set_style_nobox()

Short for style.use('nobox')

static use(*style*)

Set the style of the current figure/visualization.

Parameters **style** – matplotlib style name, or dict with values, or a sequence of these, where the last value overrides previous

Returns

9.6 ipyvolume.widgets

Test `pythreejs.Camera`

The widgets module of ipvyolume.

`ipyvolume.widgets.`**`quickvolshow`**(*data, lighting=False, data_min=None, data_max=None, max_shape=256, level=[0.1, 0.5, 0.9], opacity=[0.01, 0.05, 0.1], level_width=0.1, extent=None, memorder='C', **kwargs*)

Visualize a 3d array using volume rendering.

Parameters

- **`data`** – 3d numpy array
- **`lighting`** – boolean, to use lighting or not, if set to false, lighting parameters will be overriden
- **`data_min`** – minimum value to consider for data, if None, computed using np.nanmin
- **`data_max`** – maximum value to consider for data, if None, computed using np.nanmax
- **`max_shape`** (*`int`*) – maximum shape for the 3d cube, if larger, the data is reduced by skipping/slicing (data[::N]), set to None to disable.
- **`extent`** – list of [[xmin, xmax], [ymin, ymax], [zmin, zmax]] values that define the bounds of the volume, otherwise the viewport is used
- **`level`** – level(s) for the where the opacity in the volume peaks, maximum sequence of length 3
- **`opacity`** – opacity(ies) for each level, scalar or sequence of max length 3
- **`level_width`** – width of the (gaussian) bumps where the opacity peaks, scalar or sequence of max length 3
- **`kwargs`** – extra argument passed to Volume and default transfer function

Returns

`ipyvolume.widgets.`**`quickscatter`**(*x, y, z, **kwargs*)

`ipyvolume.widgets.`**`quickquiver`**(*x, y, z, u, v, w, **kwargs*)

`ipyvolume.widgets.`**`volshow`**(**args, **kwargs*)

Deprecated: please use ipyvolume.quickvolshow or use the ipyvolume.pylab interface.

`class` `ipyvolume.widgets.`**`Figure`**(***kwargs*)

Bases: `ipywebrtc.webrtc.MediaStream`

Widget class representing a volume (rendering) using three.js.

`ambient_coefficient`

A float trait.

`animation`

A float trait.

`animation_exponent`

A float trait.

`box_center`
An instance of a Python list.

`box_size`
An instance of a Python list.

`camera`
A `pythreejs.Camera` instance to control the camera

`camera_center`
An instance of a Python list.

`camera_control`
A trait for unicode strings.

`camera_fov`
A casting version of the float trait.

`capture_fps`
A casting version of the float trait.

`controls`
A `pythreejs.Controls` instance to control the camera

`cube_resolution`
A casting version of the int trait.

`diffuse_coefficient`
A float trait.

`displayscale`
A casting version of the float trait.

`eye_separation`
A casting version of the float trait.

`height`
A casting version of the int trait.

`lights`
An instance of a Python list.

`matrix_projection`
An instance of a Python list.

`matrix_world`
An instance of a Python list.

`meshes`
An instance of a Python list.

`mouse_mode`
A trait for unicode strings.

`on_screenshot`(*callback*, *remove=False*)

`on_selection`(*callback*, *remove=False*)

`orientation_control`
A boolean (True, False) trait.

`panorama_mode`
An enum whose value must be in a given sequence.

pixel_ratio
Pixel ratio of the WebGL canvas (2 on retina screens). Set to 1 for better performance, but less crispedges. If set to None it will use the browser's window.devicePixelRatio.

popup_debouce
Debouce popup in miliseconds

project (*x*, *y*, *z*)

render_continuous
A boolean (True, False) trait.

scales
An instance of a Python dict.

One or more traits can be passed to the constructor to validate the keys and/or values of the dict. If you need more detailed validation, you may use a custom validator method.

Changed in version 5.0: Added key_trait for validating dict keys.

Changed in version 5.0: Deprecated ambiguous `trait`, `traits` args in favor of `value_trait`, `per_key_traits`.

scatters
An instance of a Python list.

scene
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

screenshot (*width=None*, *height=None*, *mime_type='image/png'*)

selection_mode
A trait for unicode strings.

selector
A trait for unicode strings.

show
A trait for unicode strings.

slice_x
A float trait.

slice_y
A float trait.

slice_z
A float trait.

specular_coefficient
A float trait.

specular_exponent
A float trait.

stereo
A boolean (True, False) trait.

style
An instance of a Python dict.

One or more traits can be passed to the constructor to validate the keys and/or values of the dict. If you need more detailed validation, you may use a custom validator method.

Changed in version 5.0: Added key_trait for validating dict keys.

Changed in version 5.0: Deprecated ambiguous `trait`, `traits` args in favor of `value_trait`, `per_key_traits`.

volumes
An instance of a Python list.

width
A casting version of the int trait.

xlabel
A trait for unicode strings.

xlim

ylabel
A trait for unicode strings.

ylim

zlabel
A trait for unicode strings.

zlim

class ipyvolume.widgets.**Volume**(***kwargs*)
Bases: `ipywidgets.widgets.widget.Widget`, `ipyvolume.widgets.LegendData`

Widget class representing a volume (rendering) using three.js.

brightness
A casting version of the float trait.

clamp_max
A casting version of the boolean trait.

clamp_min
A casting version of the boolean trait.

data
A numpy array trait type.

data_max
A casting version of the float trait.

data_max_shape
A casting version of the int trait.

data_min
A casting version of the float trait.

data_original
A numpy array trait type.

extent
A trait which allows any value.

extent_original
A trait which allows any value.

icon
A trait for unicode strings.

lighting
A boolean (True, False) trait.

material
A `pythreejs.MeshPhongMaterial` that is used for the shading of the volume

opacity_scale
A casting version of the float trait.

ray_steps
defines the length of the ray (1/ray_steps) for each step, in normalized coordintes.

rendering_method
An enum whose value must be in a given sequence.

show_max
A casting version of the float trait.

show_min
A casting version of the float trait.

tf
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

update_data(*change=None*)

visible
A casting version of the boolean trait.

class ipyvolume.widgets.**Scatter**(***kwargs*)
Bases: `ipywidgets.widgets.widget.Widget`, `ipyvolume.widgets.LegendData`

aux
A numpy array trait type.

aux_scale
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

cast_shadow
A casting version of the boolean trait.

clicked
A boolean (True, False) trait.

clicked_index
An int trait.

color
A numpy array trait type.

color_scale
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

`color_selected`
A trait type representing a Union type.

`connected`
A casting version of the boolean trait.

`geo`
A trait for unicode strings.

`geo_matrix`
A trait for a 16-tuple corresponding to a three.js Matrix4.

`hovered`
A boolean (True, False) trait.

`hovered_index`
An int trait.

`line_material`
A `pythreejs.Material` that is used for the lines/wireframe

`material`
A `pythreejs.Material` that is used for the mesh

`popup`
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

`receive_shadow`
A casting version of the boolean trait.

`selected`
A numpy array trait type.

`sequence_index`
An int trait.

`shader_snippets`
An instance of a Python dict.

One or more traits can be passed to the constructor to validate the keys and/or values of the dict. If you need more detailed validation, you may use a custom validator method.

Changed in version 5.0: Added key_trait for validating dict keys.

Changed in version 5.0: Deprecated ambiguous `trait`, `traits` args in favor of `value_trait`, `per_key_traits`.

`size`
A trait type representing a Union type.

`size_selected`
A trait type representing a Union type.

`size_x_scale`
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

size_y_scale
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

size_z_scale
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

texture
A trait type representing a Union type.

visible
A casting version of the boolean trait.

vx
A numpy array trait type.

vy
A numpy array trait type.

vz
A numpy array trait type.

x
A numpy array trait type.

y
A numpy array trait type.

z
A numpy array trait type.

class `ipyvolume.widgets.`**Mesh**(***kwargs*)
Bases: `ipywidgets.widgets.widget.Widget`, `ipyvolume.widgets.LegendData`

cast_shadow
A casting version of the boolean trait.

clicked
A boolean (True, False) trait.

color
A numpy array trait type.

color_scale
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

hovered
A boolean (True, False) trait.

icon
A trait for unicode strings.

line_material
A `pythreejs.Material` that is used for the lines/wireframe

lines
A numpy array trait type.

material
A `pythreejs.Material` that is used for the mesh

popup
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

receive_shadow
A casting version of the boolean trait.

sequence_index
An int trait.

texture
A trait type representing a Union type.

triangles
A numpy array trait type.

u
A numpy array trait type.

v
A numpy array trait type.

visible
A casting version of the boolean trait.

volume
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

x
A numpy array trait type.

x_offset
A float trait.

y
A numpy array trait type.

y_offset
A float trait.

z
A numpy array trait type.

z_offset
A float trait.

9.7 ipyvolume.examples

Some examples for quick testing/demonstrations.

All function accept *show* and *draw* arguments

- If *draw* is *True* it will return the widgets (Scatter, Volume, Mesh)
- If *draw* is *False*, it will return the data
- if *show* is *False*, *ipv.show()* will not be called.

`ipyvolume.examples.`**`ball`**(*rmax=3, rmin=0, shape=128, limits=[-4, 4], draw=True, show=True, **kwargs*)

Show a ball.

`ipyvolume.examples.`**`brain`**(*draw=True*, *show=True*, *fiducial=True*, *flat=True*, *inflated=True*, *subject='S1'*, *interval=1000*, *uv=True*, *color=None*)

Show a human brain model.

Requirement:

$ pip install https://github.com/gallantlab/pycortex

`ipyvolume.examples.`**`example_ylm`**(*m=0, n=2, shape=128, limits=[-4, 4], draw=True, show=True, **kwargs*)

Show a spherical harmonic.

`ipyvolume.examples.`**`gaussian`**(*N=1000*, *draw=True*, *show=True*, *seed=42*, *color=None*, *marker='sphere'*, *description='Gaussian blob'*, ***kwargs*)

Show N random gaussian distributed points using a scatter plot.

`ipyvolume.examples.`**`head`**(*draw=True*, *show=True*, *max_shape=256*, *description='Male head'*)

Show a volumetric rendering of a human male head.

`ipyvolume.examples.`**`klein_bottle`**(*draw=True*, *show=True*, *figure8=False*, *endpoint=True*, *uv=True*, *wireframe=False*, *texture=None*, *both=False*, *interval=1000*, ***kwargs*)

Show one or two Klein bottles.

`ipyvolume.examples.`**`xyz`**(*shape=128, limits=[-3, 3], spherical=False, sparse=True, centers=False*)

9.8 ipyvolume.headless

CHAPTER 10

Virtual reality

Ipyvolume can render in stereo, and go fullscreen (not supported for iOS). Together with `Google Cardboard <https://vr.google.com/cardboard/>_` or other VR glasses (I am using VR Box 2) this enables virtual reality visualisation. Since mobile devices are usually less powerful, the example below is rendered at low resolution to enable a reasonable framerate on all devices.

Open this page on your mobile device, enter fullscreen mode and put on your glasses, looking around will rotate the object to improve depth perception.

```
[1]: import ipyvolume as ipv
aqa2 = ipv.datasets.aquariusA2.fetch()
fig = ipv.figure(width=256, height=256)
fig.orientation_control = True  # listen to the device changing orientation when you␣
↪move your head
ipv.volshow(aqa2.data.T, lighting=True, level=[0.16, 0.25, 0.46],
                 stereo=True, opacity=0.06)
ipv.show()
```

```
Downloading https://github.com/maartenbreddels/ipyvolume/raw/master/datasets/aquarius-
↪A2.npy.bz2 to /home/docs/.ipyvolume/datasets/aquarius-A2.npy.bz2
```

```
--2021-11-01 12:39:12--  https://github.com/maartenbreddels/ipyvolume/raw/master/
↪datasets/aquarius-A2.npy.bz2
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/maartenbreddels/ipyvolume/master/datasets/
↪aquarius-A2.npy.bz2 [following]
--2021-11-01 12:39:13--  https://raw.githubusercontent.com/maartenbreddels/ipyvolume/
↪master/datasets/aquarius-A2.npy.bz2
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133,␣
↪185.199.109.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:
↪443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 756826 (739K) [application/octet-stream]
```

(continues on next page)

(continued from previous page)

```
Saving to: ‘/home/docs/.ipyvolume/datasets/aquarius-A2.npy.bz2’

aquarius-A2.npy.bz2 100%[===================>] 739.09K  --.-KB/s    in 0.05s

2021-11-01 12:39:13 (14.7 MB/s) - ‘/home/docs/.ipyvolume/datasets/aquarius-A2.npy.
↪bz2’ saved [756826/756826]
```

```
Container(children=[VBox(children=(HBox(children=(Label(value='levels:'),␣
↪FloatSlider(value=0.16, max=1.0, ste...
```

CHAPTER 11

Integration with pythreejs

ipyvolume uses parts of pythreejs, giving a lot of flexibility to tweak the visualizations or behaviour. ## Materials The Scatter object has a `material` and `line_material` object, which both are a ShaderMaterial pythreejs object: `https://pythreejs.readthedocs.io/en/stable/api/materials/ShaderMaterial_autogen.html`.

```
[1]: import ipywidgets as widgets
     import numpy as np
     import ipyvolume as ipv
```

```
[2]: # a scatter plot
     x, y, z = np.random.normal(size=(3, 100))
     fig = ipv.figure()
     scatter = ipv.scatter(x, y, z, marker='box')
     scatter.connected = True  # draw connecting lines
     ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

Using `scatter.material` we can tweak the material setting:

```
[3]: scatter.material.visible = False
```

Or even connect a toggle button to a `line_material` property.

```
[4]: toggle_lines = widgets.ToggleButton(description="Show lines")
     widgets.jslink((scatter.line_material, 'visible'), (toggle_lines, 'value'))
     toggle_lines
```

```
ToggleButton(value=False, description='Show lines')
```

11.1 Controls

ipyvolume has builtin controls. For more flexibility, a Controls class from https://pythreejs.readthedocs.io/en/stable/api/controls/index.html can be contructed.

```
[5]: import pythreejs
import ipyvolume as ipv
import numpy as np
fig = ipv.figure()
scatter = ipv.scatter(x, y, z, marker='box')
ipv.show()

control = pythreejs.OrbitControls(controlling=fig.camera)
# assigning to fig.controls will overwrite the builtin controls
fig.controls = control
control.autoRotate = True
# the controls does not update itself, but if we toggle this setting, ipyvolume will␣
↪update the controls
fig.render_continuous = True
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

```
/home/docs/checkouts/readthedocs.org/user_builds/ipyvolume/envs/latest/lib/python3.7/
↪site-packages/jupyter_client/session.py:716: UserWarning: Message serialization␣
↪failed with:
Out of range float values are not JSON compliant
Supporting this message is deprecated in jupyter-client 7, please make sure your␣
↪message is JSON-compliant
  content = self.pack(content)
```

```
[6]: control.autoRotate = True
toggle_rotate = widgets.ToggleButton(description="Rotate")
widgets.jslink((control, 'autoRotate'), (toggle_rotate, 'value'))
toggle_rotate
```

```
ToggleButton(value=False, description='Rotate')
```

11.2 Camera

The camera property of ipyvolume is by default a PerspectiveCamera, but other cameras should also work: https://pythreejs.readthedocs.io/en/stable/api/cameras/index.html

```
[7]: text = widgets.Text()
widgets.jslink((fig.camera, 'position'), (text, 'value'))
text
```

```
Text(value='')
```

CHAPTER 12

Ipyvolume

IPyvolume is a Python library to visualize 3d volumes and glyphs (e.g. 3d scatter plots), in the Jupyter notebook, with minimal configuration and effort. It is currently pre-1.0, so use at own risk. IPyvolume's *volshow* is to 3d arrays what matplotlib's imshow is to 2d arrays.

Other (more mature but possibly more difficult to use) related packages are yt, VTK and/or Mayavi.

Feedback and contributions are welcome: Github, Email or Twitter. —

12.1 Quick intro

12.1.1 Volume

For quick resuls, use `ipyvolume.widgets.quickvolshow`. From a numpy array, we create two boxes, using slicing, and visualize it.

```
[1]: import numpy as np
     import ipyvolume as ipv
     V = np.zeros((128,128,128)) # our 3d array
     # outer box
     V[30:-30,30:-30,30:-30] = 0.75
     V[35:-35,35:-35,35:-35] = 0.0
     # inner box
     V[50:-50,50:-50,50:-50] = 0.25
     V[55:-55,55:-55,55:-55] = 0.0
     ipv.quickvolshow(V, level=[0.25, 0.75], opacity=0.03, level_width=0.1, data_min=0,␣
     ↪data_max=1)
```

```
Container(children=[VBox(children=(HBox(children=(Label(value='levels:'),␣
↪FloatSlider(value=0.25, max=1.0, ste...
```

12.1.2 Scatter plot

Simple scatter plots are also supported.

[2]:
```
import ipyvolume as ipv
import numpy as np
x, y, z = np.random.random((3, 10000))
ipv.quickscatter(x, y, z, size=1, marker="sphere")
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

12.1.3 Quiver plot

Quiver plots are also supported, showing a vector at each point.

[3]:
```
import ipyvolume as ipv
import numpy as np
x, y, z, u, v, w = np.random.random((6, 1000))*2-1
ipv.quickquiver(x, y, z, u, v, w, size=5)
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

12.1.4 Mesh plot

And surface/mesh plots, showing surfaces or wireframes.

[4]:
```
import ipyvolume as ipv
x, y, z, u, v = ipv.examples.klein_bottle(draw=False)
ipv.figure()
m = ipv.plot_mesh(x, y, z, wireframe=False)
ipv.squarelim()
ipv.show()
```

```
Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0, 1.0],␣
↪camera=PerspectiveCamera(fov=45....
```

12.1.5 Built on Ipywidgets

For anything more sophisticated, use `ipyvolume.pylab`, ipyvolume's copy of matplotlib's 3d plotting (+ volume rendering).

Since ipyvolume is built on ipywidgets, we can link widget's properties.

[5]:
```
import ipyvolume as ipv
import numpy as np
x, y, z, u, v, w = np.random.random((6, 1000))*2-1
selected = np.random.randint(0, 1000, 100)
ipv.figure()
quiver = ipv.quiver(x, y, z, u, v, w, size=5, size_selected=8, selected=selected)

from ipywidgets import FloatSlider, ColorPicker, VBox, jslink
size = FloatSlider(min=0, max=30, step=0.1)
```

(continues on next page)

(continued from previous page)

```
size_selected = FloatSlider(min=0, max=30, step=0.1)
color = ColorPicker()
color_selected = ColorPicker()
jslink((quiver, 'size'), (size, 'value'))
jslink((quiver, 'size_selected'), (size_selected, 'value'))
jslink((quiver, 'color'), (color, 'value'))
jslink((quiver, 'color_selected'), (color_selected, 'value'))
VBox([ipv.gcc(), size, size_selected, color, color_selected])
```

```
VBox(children=(Container(figure=Figure(box_center=[0.5, 0.5, 0.5], box_size=[1.0, 1.0,
↪ 1.0], camera=Perspectiv...
```

Try changing the slider to the change the size of the vectors, or the colors.

12.2 Quick installation

This will most likely work, otherwise read *install*

```
pip install ipyvolume
jupyter nbextension enable --py --sys-prefix ipyvolume
jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

For conda/anaconda, use:

conda install -c conda-forge ipyvolume

12.3 About

Ipyvolume is an offspring project from vaex. Ipyvolume makes use of threejs, an excellent Javascript library for OpenGL/WebGL rendering.

12.4 Contents

install examples api vr pythreejs

12.5 Changelog

- 0.5
  - New
    - Volume is now its own widget, allowing multivolume rendering

  * Depth aware zooming (Hold Alt key, or toggle in menu) and zoom into any object or volume rendering

    - double click centers that point
    - Configurable ray steps for volume rendering (Volume.ray_steps)
    - Better transparency support, premultiplied colors used consistently, colors can now be of shape $(..., 3or4)$ to allow alpha channel (note: no proper rendering yet, this is a difficult problem).

        * Panoramic modes: 180 and 360 degrees for dome projection or VR video creations.
        * Maximum intensity volume rendering.
        * Progressive loading of large volumetric cubes.
        * ipyvolume.moviemaker: simple UI for making movies, and keyframes settings for the camera.
        * ipyvolume.astro: (experiment) as domain specific module for astronomy.
        * New example male head volume rendering `ipyvolume.examples.head`
    - Changes
        * 100x faster mesh generation
        * Fixes/improvements for headless rendering
        * Selection method in the kernel, see `ipyvolume.pylab.selector_default`.
        * Fixed memory leak issues in the browser
        * Scatter supports 2d sprites,see `ipyvolume.pylab.scatter`.
        * Pythreejs integration, Camera, Scene and ShaderMaterial are now exposed.
        * 'sphere' marker was double the size as the others, now halved in size/
        * `ipyvolume.pylab.view` can control distance, and returns currents values.
    - New contributors
        * Casper van Leeuwen
        * Oleh Kozynets
        * Oliver Evans
        * Jean-Rémi KING
        * Mathieu Carette
        * Saul (saulthu)
        * Timo Friedri
        * WANG Aiyong
        * mpu-creare
        * xavArtley
        * Eric Larson
        * Hans Moritz Günther
        * Jackie Leng
- 0.4
    - plotting
        * lines
        * wireframes
        * meshes/surfaces
        * isosurfaces
        * texture (animated) support, gif image and MediaStream (movie, camera, canvas)

  - camera control (angles from the python side), FoV
  - movie creation
  - eye separation for VR
  - better screenshot support (can be to a PIL Image), and higher resolution possible
  - mouse lasso (a bit rough), selections can be made from the Python side.
  - icon bar for common operations (fullscreen, stereo, screenshot, reset etc)
  - offline support for embedding/saving to html
  - Jupyter lab support
  - New contributors
    * Chris Sewell
    * Satrajit Ghosh
    * Sylvain Corlay
    * stonebig
    * Matt McCormick
    * Jean Michel Arbona
- 0.3
  - new
    * axis with labels and ticklabels
    * styling
    * animation (credits also to https://github.com/jeammimi)
    * binary transfers
    * default camera control is trackball
  - changed
    * s and ss are now spelled out, size and size_selected

Python Module Index

Index