plotly

Star 23,446

Dash Python > ***Overview & Reference***

Plotly Studio: Transform any dataset into an interactive data application in minutes with AI. **Sign up for early access now.**

Introduction to Dash Slicer

The `dash_slicer` library provides an easy way to visualize 3D image data by slicing along one dimension. Multiple views on the same data can be linked, to help with navigation. There is also support for mask overlays.

Install Dash Slicer with:

`pip install -U dash-slicer`

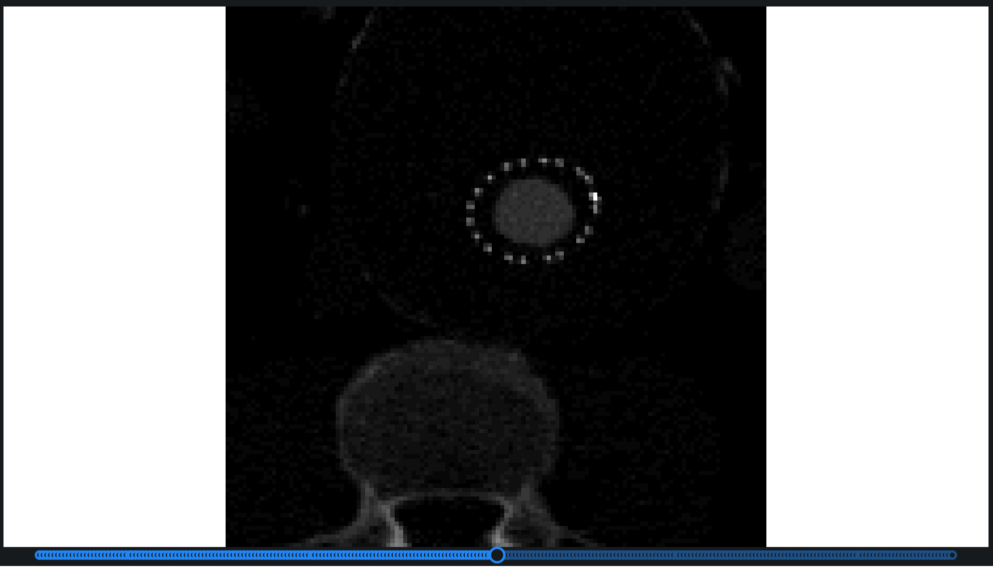
The source is on GitHub at **plotly/dash-slicer**.

Guide

A first example

Let's get started with a simple example. (The examples on this page are built with dash-slicer version 0.3.0)

```python
from dash import Dash, html
import imageio
from dash_slicer import VolumeSlicer


app = Dash(__name__, update_title=None)

vol = imageio.volread("imageio:stent.npz")
slicer = VolumeSlicer(app, vol)
slicer.graph.config["scrollZoom"] = False

app.layout = html.Div([slicer.graph, slicer.slider, *slicer.stores])


if __name__ == "__main__":
    app.run(debug=True, dev_tools_props_check=False)
```



[illegible]

In the code above, we first load a 3D numpy array (a volumetric image). Then we instantiate a `VolumeSlicer` object with that data. This object is not a Dash component, but has attributes that are. We place its `graph` and `slider` in the layout, as well as a handful of `Store` components that the slicer needs to function.

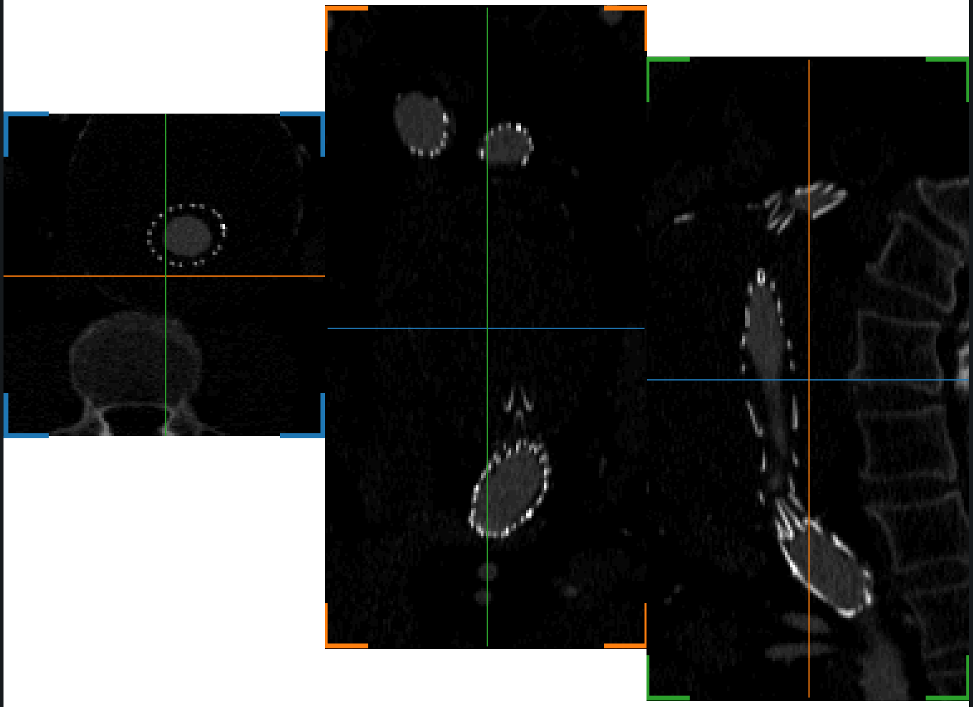
The graph's `scrollZoom` config option is turned off here, because it would make the plot grab scroll events as you scroll down this page.

If the server is run in debug mode, consider setting `dev_tools_props_check` to False, because it has a big impact on the performance.

Multiple slicers

In the next example, we create multiple slicers, one for each dimension, and put them in a layout, just like we did in the previous example.

```python
from dash import Dash, html
import imageio
from dash_slicer import VolumeSlicer


app = Dash(__name__, update_title=None)

vol = imageio.volread("imageio:stent.npz")
slicer0 = VolumeSlicer(app, vol, axis=0)
slicer1 = VolumeSlicer(app, vol, axis=1)
slicer2 = VolumeSlicer(app, vol, axis=2)

slicer0.graph.config["scrollZoom"] = False
slicer1.graph.config["scrollZoom"] = False
slicer2.graph.config["scrollZoom"] = False

app.layout = html.Div(
    style={
        "display": "grid",
        "gridTemplateColumns": "33% 33% 33%",
    },
    children=[
        html.Div([slicer0.graph, html.Br(), slicer0.slider, *slicer0.stores]),
        html.Div([slicer1.graph, html.Br(), slicer1.slider, *slicer1.stores]),
        html.Div([slicer2.graph, html.Br(), slicer2.slider, *slicer2.stores]),
    ],
)


if __name__ == "__main__":
    app.run(debug=True, dev_tools_props_check=False)
```



You can see how each view contains indicator lines, which show the position of the other slicers. Each slicer has a certain color (which is auto-selected, but can also be provided), which is shown in the corners around the image. The same color is used to draw the slicer's indicator lines in the other views.

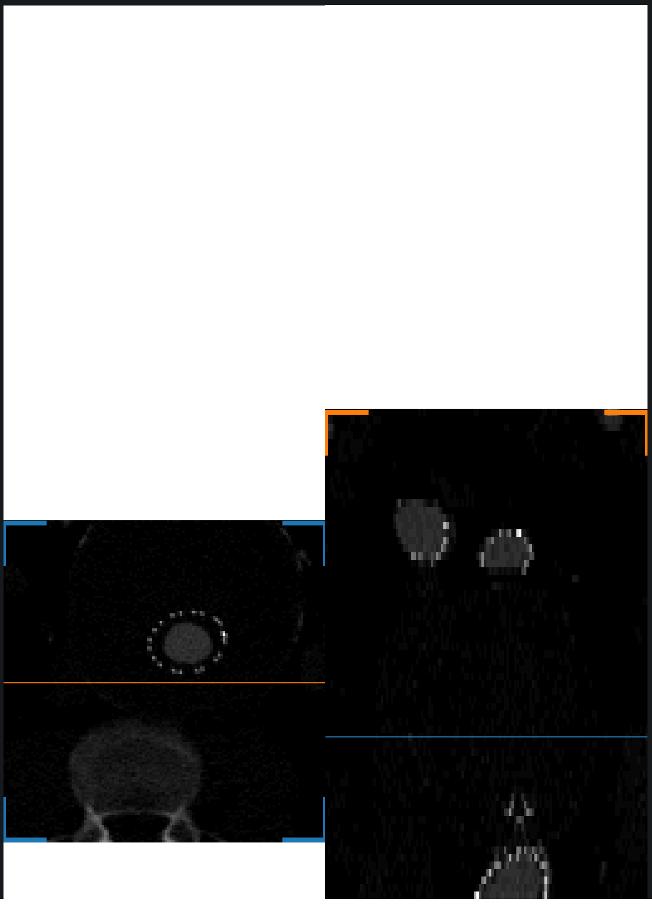
Slicers indicate each-other's position if they are linked based on their `scene_id`. This is a property that can be provided when you instantiate a `VolumeSlicer`. By default, the `scene_id` is derived from the volume. That's why the linking in this example works: each slicer is given the same numpy array object. By explicitly setting the `scene_id`, multiple views on different data (e.g. MRI and CT) can be linked as well.

In addition to using the sliders, you can click in one of the views to make the other views go to the clicked location. Try it! Thanks to this navigation mode, you can optionally hide sliders in the layout when two or more views are present (see the `VolumeSlicer.slider` property).

Anisotropic data

In the next example, we make the data non-isotropic. This means that the distance between voxels is not equal for all dimensions. The voxel-spacing can be provided via the `spacing` argument. Similarly, an `origin` can also be provided. You can zoom into the view on the right to see that the voxels are elongated.

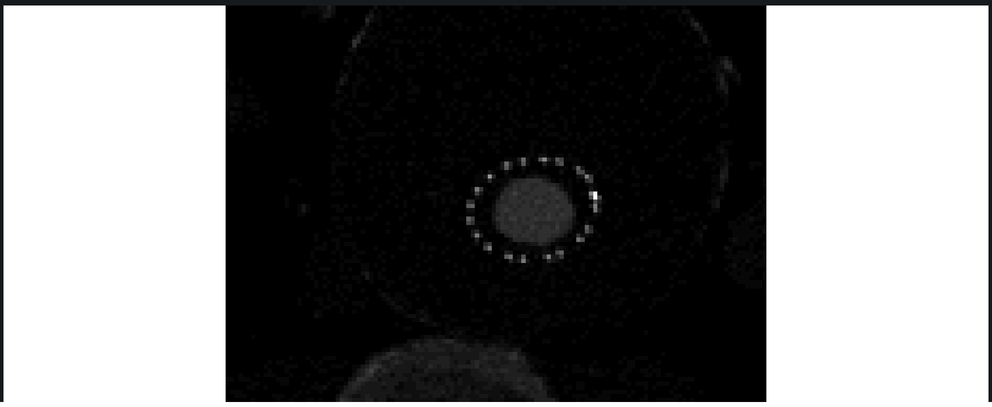
```python
from dash import Dash, html
import imageio
from dash_slicer import VolumeSlicer


app = Dash(__name__, update_title=None)

vol = imageio.volread("imageio:stent.npz")
vol = vol[::3,:,:]
spacing = 3, 1, 1

slicer0 = VolumeSlicer(app, vol, spacing=spacing, axis=0)
slicer1 = VolumeSlicer(app, vol, spacing=spacing, axis=1)

slicer0.graph.config["scrollZoom"] = False
slicer1.graph.config["scrollZoom"] = False

app.layout = html.Div(
    style={
        "display": "grid",
        "gridTemplateColumns": "33% 33%",
    },
    children=[
        html.Div([slicer0.graph, html.Br(), slicer0.slider, *slicer0.stores]),
        html.Div([slicer1.graph, html.Br(), slicer1.slider, *slicer1.stores]),
    ],
)


if __name__ == "__main__":
    app.run(debug=True, dev_tools_props_check=False)
```

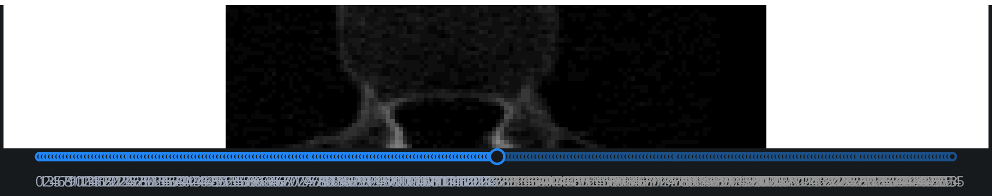


Reacting to the slicer state

This example illustrates how your application can react to the slicer's position and view by using the `state` store as an input. Note that the id of this store is a dict, which makes it possible to write a **pattern matching Input** to collect the states of all slicers with a certain scene_id. See the reference docs for details.

```
import json
from dash import Dash, html, Input, Output, callback
import imageio
from dash_slicer import VolumeSlicer


app = Dash(__name__, update_title=None)

vol = imageio.volread("imageio:stent.npz")
slicer = VolumeSlicer(app, vol)
slicer.graph.config["scrollZoom"] = False

app.layout = html.Div([slicer.graph, slicer.slider, html.Div(id='info'), *slicer.stores])

@callback(
    Output("info", "children"),
    Input(slicer.state.id, "data")
)
def update_info(state):
    return json.dumps(state, indent=4)


if __name__ == "__main__":
    app.run(debug=True, dev_tools_props_check=False)
```





```
{ "index": 128, "index_changed": true, "xrange": [ -0.5, 127.5 ], "yrange": [ 0.06888888888888889,
126.93363950617284 ], "zpos": 128, "axis": 0, "color": "#1F77B4" }
```

More examples

More examples are available at the **dash-slicer repository**.

Dash Python ***> Overview & Reference***