ipygany

QuantStack

Jan 26, 2021

INSTALLATION

CHAPTER

ONE

USING CONDA

```
conda install -c conda-forge ipygany
```

If you want to load vtk files, it will be needed to install the vtk library, you can install it from conda-forge as well:

```
conda install -c conda-forge vtk
```

CHAPTER

TWO

USING PIP

```
pip install ipygany
jupyter nbextension enable --py --sys-prefix ipygany  # can be skipped for notebook 5.
↪3 and above
```

If you want to load vtk files, it will be needed to install the vtk library.

CHAPTER

THREE

JUPYTERLAB EXTENSION

If you have JupyterLab, you will also need to install the JupyterLab extension:

```
jupyter labextension install @jupyter-widgets/jupyterlab-manager ipygany
```

CHAPTER

FOUR

DEVELOPMENT INSTALLATION

For a development installation (requires npm):

```
git clone https://github.com/jupyter-widgets/ipygany.git
cd ipygany
pip install -e .
jupyter nbextension install --py --symlink --sys-prefix ipygany
jupyter nbextension enable --py --sys-prefix ipygany
jupyter labextension install @jupyter-widgets/jupyterlab-manager .  # If you are␣
↪developing on JupyterLab
```

CHAPTER

FIVE

USAGE

`ipygany` provides a set of tools for loading and analyzing 3D meshes in the Jupyter Notebook.

5.1 Loading your mesh in the Notebook

`ipygany` provides a `PolyMesh` class for loading triangle-based meshes, and a `TetraMesh` class for tetrahedron-based meshes.

You can either use vtk to load your meshes, or manually pass vertices buffers.

You need to create a 3D `Scene` widget in order to display your loaded mesh in the page.

```
from ipygany import Scene, PolyMesh

mesh = PolyMesh.from_vtk('assets/fastscapelib_topo.vtk')
mesh.default_color = 'gray'

scene = Scene([mesh])
scene
```

5.2 Applying effects to your mesh

You can apply multiple effects to your mesh for quick analysis, most of those effects are performed on the GPU, hence are super fast.

```
from ipygany import IsoColor, Warp

colored_mesh = IsoColor(mesh, input='H', min=0., max=1003.)
warped_mesh = Warp(colored_mesh, input=(0, 0, ('H', 'X1')), warp_factor=0.)

scene2 = Scene([warped_mesh])
scene2
```

You can then control some parameters using other widgets:

```
from ipywidgets import FloatSlider, jslink

warp_slider = FloatSlider(value=0., min=0., max=3.)

jslink((warped_mesh, 'factor'), (warp_slider, 'value'))
```

(continues on next page)

(continued from previous page)

```
warp_slider
```

CHAPTER

SIX

POLYMESH

The `PolyMesh` widget represents a triangle-based unstructured mesh.

You can either load a mesh from a `vtk` file (needs the vtk library installed), or manually populate the vertices and triangle data.

6.1 Example

Load from a `vtk` file:

```
from ipygany import Scene, PolyMesh


mesh = PolyMesh.from_vtk('assets/fastscapelib_topo.vtk')

scene = Scene([mesh])
scene
```

Load from memory:

```
import numpy as np


# Create triangle indices
nx = 100
ny = 100

triangle_indices = np.empty((ny - 1, nx - 1, 2, 3), dtype=int)

r = np.arange(nx * ny).reshape(ny, nx)

triangle_indices[:, :, 0, 0] = r[:-1, :-1]
triangle_indices[:, :, 1, 0] = r[:-1, 1:]
triangle_indices[:, :, 0, 1] = r[:-1, 1:]

triangle_indices[:, :, 1, 1] = r[1:, 1:]
triangle_indices[:, :, :, 2] = r[1:, :-1, None]

triangle_indices.shape = (-1, 3)

# Create vertices
x = np.arange(-5, 5, 10/nx)
y = np.arange(-5, 5, 10/ny)
```

(continues on next page)

(continued from previous page)

```
xx, yy = np.meshgrid(x, y, sparse=True)

z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)

vertices = np.empty((ny, nx, 3))
vertices[:, :, 0] = xx
vertices[:, :, 1] = yy
vertices[:, :, 2] = z
vertices = vertices.reshape(nx * ny, 3)

mesh = PolyMesh(
    vertices=vertices,
    triangle_indices=triangle_indices
)

scene = Scene([mesh])
scene
```

CHAPTER

SEVEN

TETRAMESH

The `TetraMesh` widget represents a tetrahedron-based unstructured mesh.

You can either load a mesh from a `vtk` file (needs the vtk library installed), or manually populate the vertices and tetrahedron data.

7.1 Example

Load from a `vtk` file:

```
from ipygany import Scene, TetraMesh


mesh = TetraMesh.from_vtk('assets/piston.vtu')

scene = Scene([mesh])
scene
```

Load from memory:

```
# Creating a brick with meshpy
import numpy as np
from meshpy.tet import MeshInfo, build


mesh_info = MeshInfo()
mesh_info.set_points([
    (0,0,0), (2,0,0), (2,2,0), (0,2,0),
    (0,0,12), (2,0,12), (2,2,12), (0,2,12),
])
mesh_info.set_facets([
    [0,1,2,3],
    [4,5,6,7],
    [0,4,5,1],
    [1,5,6,2],
    [2,6,7,3],
    [3,7,4,0],
])
mesh = build(mesh_info)

mesh = TetraMesh(
    vertices=np.asarray(mesh.points),
    tetrahedron_indices=np.asarray(mesh.elements)
```

(continues on next page)

(continued from previous page)

```
)

scene = Scene([mesh])
scene
```

CHAPTER

EIGHT

DATA

Before applying any effect to your mesh, you will need to load node data.

The `Data` and `Component` widgets represents the node data for an unstructured mesh.

- A `Data` widget is made of a `name` property and a list of components. As an example, a temperature `Data` (1D) would be made of a list of one `Component`, a displacement `Data` (3D) would be made of a list of three `Components`.
- A `Component` widget is made of a `name` property and an `array` property.

Those data are loaded automatically when loading a mesh from a vtk file. If you manually create a `PolyMesh` or a `TetraMesh`, you will need to manually create the node data.

8.1 Example

When loading a vtk file, the `Data` and `Component` widgets are created automatically from the node data in the file:

```
from ipygany import TetraMesh


mesh = TetraMesh.from_vtk('assets/piston.vtu')

for data in mesh.data:
    print(data.name, ':', [component.name for component in data.components])
```

```
RESU____DEPL : ['DX', 'DY', 'DZ']
RESU____SIGM_NOEU : ['SIXX', 'SIYY', 'SIZZ', 'SIXY', 'SIXZ', 'SIYZ']
```

When loading a mesh from memory, you will need to manually pass your node data:

```
import numpy as np
from ipygany import Scene, PolyMesh, Component, IsoColor


# Create triangle indices
Nr = 100
Nc = 100

triangle_indices = np.empty((Nr - 1, Nc - 1, 2, 3), dtype=int)

r = np.arange(Nr * Nc).reshape(Nr, Nc)

```

(continues on next page)

(continued from previous page)

```
triangle_indices[:, :, 0, 0] = r[:-1, :-1]
triangle_indices[:, :, 1, 0] = r[:-1, 1:]
triangle_indices[:, :, 0, 1] = r[:-1, 1:]

triangle_indices[:, :, 1, 1] = r[1:, 1:]
triangle_indices[:, :, :, 2] = r[1:, :-1, None]

triangle_indices.shape = (-1, 3)

# Create vertices
x = np.arange(-5, 5, 0.1)
y = np.arange(-5, 5, 0.1)

xx, yy = np.meshgrid(x, y, sparse=True)

z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)

vertices = np.empty((100, 100, 3))
vertices[:, :, 0] = xx
vertices[:, :, 1] = yy
vertices[:, :, 2] = z
vertices = vertices.reshape(10000, 3)

height_component = Component(name='value', array=z)

mesh = PolyMesh(
    vertices=vertices,
    triangle_indices=triangle_indices,
    data={'height': [height_component]}
)

# Colorize by curvature
colored_mesh = IsoColor(mesh, input=('height', 'value'), min=np.min(z), max=np.max(z))

scene = Scene([colored_mesh])
scene
```

CHAPTER

NINE

ISOCOLOR

The `IsoColor` widget colorize your mesh given.

Note: Currently, you can only use the default Viridis colormap, we'd like to support all paraview colormaps

The `input` attribute should be the name of the `Component` you want to use for colorizing the mesh. You also need to pass the `min` and `max` (or `range` as a min/max tuple) of the component array.

For example, if you have a 1-D `Data` named `"height"`, you can simply pass its name as input:

```
isocolor_mesh = IsoColor(mesh, input='height')
```

If you have a 3-D `Data`, you will need to pass the right component name, by passing a tuple containing (data name, component name):

```
isocolor_mesh = IsoColor(mesh, input=('displacement', 'z'))
```

9.1 Examples

```
import numpy as np
from ipywidgets import FloatSlider, FloatRangeSlider, Dropdown, Select, VBox,␣
↪AppLayout, jslink
from ipygany import Scene, IsoColor, PolyMesh, Component, ColorBar, colormaps


# Create triangle indices
nx = 100
ny = 100

triangle_indices = np.empty((ny - 1, nx - 1, 2, 3), dtype=int)

r = np.arange(nx * ny).reshape(ny, nx)

triangle_indices[:, :, 0, 0] = r[:-1, :-1]
triangle_indices[:, :, 1, 0] = r[:-1, 1:]
triangle_indices[:, :, 0, 1] = r[:-1, 1:]

triangle_indices[:, :, 1, 1] = r[1:, 1:]
triangle_indices[:, :, :, 2] = r[1:, :-1, None]

triangle_indices.shape = (-1, 3)
```

(continues on next page)

(continued from previous page)

```
# Create vertices
x = np.arange(-5, 5, 10/nx)
y = np.arange(-5, 5, 10/ny)

xx, yy = np.meshgrid(x, y, sparse=True)

z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)

vertices = np.empty((ny, nx, 3))
vertices[:, :, 0] = xx
vertices[:, :, 1] = yy
vertices[:, :, 2] = z
vertices = vertices.reshape(nx * ny, 3)

height_component = Component(name='value', array=z)

mesh = PolyMesh(
    vertices=vertices,
    triangle_indices=triangle_indices,
    data={'height': [height_component]}
)

height_min = np.min(z)
height_max = np.max(z)

# Colorize by height
colored_mesh = IsoColor(mesh, input='height', min=height_min, max=height_max)

# Create a slider that will dynamically change the boundaries of the colormap
colormap_slider_range = FloatRangeSlider(value=[height_min, height_max], min=height_
↪min, max=height_max, step=(height_max - height_min) / 100.)

jslink((colored_mesh, 'range'), (colormap_slider_range, 'value'))

# Create a colorbar widget
colorbar = ColorBar(colored_mesh)

# Colormap choice widget
colormap = Dropdown(
    options=colormaps,
    description='colormap:'
)

jslink((colored_mesh, 'colormap'), (colormap, 'index'))


AppLayout(
    left_sidebar=Scene([colored_mesh]),
    right_sidebar=VBox((colormap_slider_range, colormap, colorbar)),
    pane_widths=[2, 0, 1]
)
```

CHAPTER

TEN

WARP

The `Warp` widget will modify the mesh geometry.

It is similar to `Paraview`, `PyVista` and `vtk`'s warp-by-vector effect, but instead of computing the transformation on the CPU, it is entirely computed on the GPU. Which means that changing the warp factor does not involve looping over the mesh vertices, we only send the new `factor` value to the GPU.

The `input` attribute should be a 3-D tuple containing `Components` names or floating point values. For example, if your mesh has a 3-D `Data` named `"displacement"`, your can set the input to be:

```
warped_mesh = Warp(mesh, input='displacement')
```

If you only want to visualize the last component of your displacement data, simply set the other two components to `0`:

```
warped_mesh = Warp(mesh, input=(0, 0, ('displacement', 'z')))
```

If your mesh contains a 1-D `Data`, you can also warp using this data by setting it to the wanted dimension:

```
x_warped_mesh = Warp(mesh, input=('height', 0, 0))  # Warp by 'height' data on the x-
↪axis
z_warped_mesh = Warp(mesh, input=(0, 0, 'height'))  # Warp by 'height' data on the z-
↪axis
```

10.1 Examples

```
from ipywidgets import FloatSlider, VBox, jslink
from ipygany import Scene, IsoColor, TetraMesh, Warp

# Load a Piston mesh, which contains displacement data
mesh = TetraMesh.from_vtk('assets/piston.vtu')

# Colorize the mesh by the dX displacement
colored_mesh = IsoColor(mesh, input=('RESU____DEPL', 'DX'), min=-1.39e-06, max=1.39e-
↪06)

# Warp by the displacement data (dX, dY, dZ)
warped_mesh = Warp(colored_mesh, input='RESU____DEPL', factor=300)

# Create a slider that will dynamically change the warp factor value
warp_slider = FloatSlider(value=300, min=0, max=800)

jslink((warped_mesh, 'factor'), (warp_slider, 'value'))
```

(continues on next page)

(continued from previous page)

```
VBox((Scene([warped_mesh]), warp_slider))
```

You can also combine it with other effects like `Threshold`:

Like other ipygany's effects, you can combine it with other effects. Here we applied an `IsoColor` effect, followed by a `Warp` effect, and we finally apply a `Threshold` effect that will hide parts of the mesh where `dX` [-1.39e-06, 1.0e-07]:

```
from ipywidgets import FloatRangeSlider
from ipygany import Threshold


threshold_mesh = Threshold(warped_mesh, input=('RESU____DEPL', 'DX'), min=-1.39e-06,␣
↪max=1.0e-07)

VBox((Scene([threshold_mesh]), warp_slider))
```

```
import numpy as np
from ipygany import Scene, PolyMesh, Component, IsoColor


# Create triangle indices
nx = 100
ny = 100

triangle_indices = np.empty((ny - 1, nx - 1, 2, 3), dtype=int)

r = np.arange(nx * ny).reshape(ny, nx)

triangle_indices[:, :, 0, 0] = r[:-1, :-1]
triangle_indices[:, :, 1, 0] = r[:-1, 1:]
triangle_indices[:, :, 0, 1] = r[:-1, 1:]

triangle_indices[:, :, 1, 1] = r[1:, 1:]
triangle_indices[:, :, :, 2] = r[1:, :-1, None]

triangle_indices.shape = (-1, 3)

# Create vertices
x = np.arange(-5, 5, 10/nx)
y = np.arange(-5, 5, 10/ny)

xx, yy = np.meshgrid(x, y, sparse=True)

z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)

vertices = np.empty((ny, nx, 3))
vertices[:, :, 0] = xx
vertices[:, :, 1] = yy
vertices[:, :, 2] = z
vertices = vertices.reshape(nx * ny, 3)

height_component = Component(name='value', array=z)

mesh = PolyMesh(
```

(continues on next page)

(continued from previous page)

```
    vertices=vertices,
    triangle_indices=triangle_indices,
    data={'height': [height_component]}
)

# Colorize by curvature
colored_mesh = IsoColor(mesh, input='height', min=np.min(z), max=np.max(z))
warped_mesh = Warp(colored_mesh, input=(0, 0, 'height'))

# Create a slider that will dynamically change the warp factor value
warp_slider = FloatSlider(value=0, min=0, max=1)

jslink((warped_mesh, 'factor'), (warp_slider, 'value'))

VBox((Scene([warped_mesh]), warp_slider))
```

CHAPTER

ELEVEN

WARPBYSCALAR

The `WarpByScalar` widget will modify the mesh geometry.

It is similar to `Paraview`, `PyVista` and `vtk`'s warp-by-scalar effect, but instead of computing the transformation on the CPU, it is entirely computed on the GPU. Which means that changing the warp factor does not involve looping over the mesh vertices, we only send the new `factor` value to the GPU.

The `input` attribute should be a 1-D data. For example, if your mesh has a 1-D `Data` named `"height"`, your can set the input to be:

```
warped_mesh = WarpByScalar(mesh, input='height')  # Warp by 'height' data
```

11.1 Examples

```
from pyvista import examples
import numpy as np

from ipywidgets import VBox, FloatSlider
from ipygany import PolyMesh, Scene, IsoColor, WarpByScalar

pvmesh = examples.download_topo_global()
ugrid = pvmesh.cast_to_unstructured_grid()

from ipygany import PolyMesh, Scene, IsoColor, WarpByScalar

# Turn the PyVista mesh into a PolyMesh
mesh = PolyMesh.from_vtk(ugrid)

colored_mesh = IsoColor(mesh, min=-10421.0, max=6527.0)
warped_mesh = WarpByScalar(colored_mesh, input='altitude', factor=0.5e-5)

# Link a slider to the warp value
warp_slider = FloatSlider(min=0., max=5., value=0.5)

def on_slider_change(change):
    warped_mesh.factor = change['new'] * 1e-5

warp_slider.observe(on_slider_change, 'value')

VBox((warp_slider, Scene([warped_mesh])))
```

CHAPTER

TWELVE

THRESHOLD

The `Threshold` widget hides part of your mesh for which the does not fit in a given range.

The `input` attribute should be the name of the `Component` you want to use for hiding the mesh. You also need to pass the `min` and `max` (or `range` as a min/max tuple) of that the component should respect.

For example, if you have a 1-D `Data` named `"height"`, you can simply pass its name as input:

```
threshold_mesh = Threshold(mesh, input='height')
```

If you have a 3-D `Data`, you will need to pass the right component name, by passing a tuple containing (data name, component name):

```
threshold_mesh = Threshold(mesh, input=('displacement', 'z'))
```

12.1 Examples

```
import numpy as np
from ipywidgets import FloatSlider, FloatRangeSlider, VBox, jslink
from ipygany import Scene, Threshold, PolyMesh, Component


# Create triangle indices
nx = 100
ny = 100

triangle_indices = np.empty((ny - 1, nx - 1, 2, 3), dtype=int)

r = np.arange(nx * ny).reshape(ny, nx)

triangle_indices[:, :, 0, 0] = r[:-1, :-1]
triangle_indices[:, :, 1, 0] = r[:-1, 1:]
triangle_indices[:, :, 0, 1] = r[:-1, 1:]

triangle_indices[:, :, 1, 1] = r[1:, 1:]
triangle_indices[:, :, :, 2] = r[1:, :-1, None]

triangle_indices.shape = (-1, 3)

# Create vertices
x = np.arange(-5, 5, 10/nx)
y = np.arange(-5, 5, 10/ny)
```

(continues on next page)

(continued from previous page)

```
xx, yy = np.meshgrid(x, y, sparse=True)

z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)

vertices = np.empty((ny, nx, 3))
vertices[:, :, 0] = xx
vertices[:, :, 1] = yy
vertices[:, :, 2] = z
vertices = vertices.reshape(nx * ny, 3)

height_component = Component(name='value', array=z)

mesh = PolyMesh(
    vertices=vertices,
    triangle_indices=triangle_indices,
    data={'height': [height_component]}
)

height_min = np.min(z)
height_max = np.max(z)

# Hide parts of the mesh
threshold_mesh = Threshold(mesh, input='height', min=height_min, max=height_max)

# Create a slider that will dynamically change the boundaries of the threshold
threshold_slider_range = FloatRangeSlider(value=[height_min, height_max], min=height_
↪min, max=height_max, step=(height_max - height_min) / 100.)

jslink((threshold_mesh, 'range'), (threshold_slider_range, 'value'))

VBox((Scene([threshold_mesh]), threshold_slider_range))
```

CHAPTER

THIRTEEN

WATER

ipygany provides a `Water` widget that displays your mesh using reflection and refraction of the environment. It also provides an `UnderWater` widget for all meshes that are under-water, ipygany will cast caustics on these under-water meshes.

Unlike most of the ipygany effects, the `Water` widget has no input. You simply create it by passing the mesh on which you want to apply the effect:

```
water_mesh = Water(mesh)
```

If you have under-water meshes on which you want to cast caustics, you need to create an `UnderWater` widget for each for them. The input must be a 1-D component specifying if the node is under-water or not.

```
underwater_mesh = UnderWater(mesh, input='underwater')
```

Then you'll need to pass those under-water meshes to the water effect:

```
underwater_mesh1 = UnderWater(mesh1, input='underwater')
underwater_mesh2 = UnderWater(mesh2, input='underwater')

water_mesh = Water(mesh, under_water_blocks=[underwater_mesh1, underwater_mesh2])
```

You can find a full example Notebook here: https://github.com/martinRenou/proteus_visualization/blob/master/reef.ipynb

CHAPTER

FOURTEEN

PYVISTA

https://docs.pyvista.org

PyVista is a great library that exposes a high level API for vtk, it has way more features than ipygany. For this reason, ipygany supports PyVista objects:

```
import pyvista as pv
from pyvista import examples

pvmesh = examples.download_st_helens()
ugrid = pvmesh.cast_to_unstructured_grid()

from ipygany import PolyMesh, Scene, IsoColor, Warp

# Turn the PyVista mesh into a PolyMesh
mesh = PolyMesh.from_vtk(ugrid)
warped_mesh = Warp(mesh, input=(0, 0, ('Elevation', 'X1')), warp_factor=1.)
colored_mesh = IsoColor(warped_mesh, input='Elevation', min=682, max=2543)

Scene([colored_mesh])
```

```
import pyvista as pv
from pyvista import examples

nefertiti = examples.download_nefertiti()
ugrid = nefertiti.cast_to_unstructured_grid()

from ipygany import PolyMesh, Scene, IsoColor, Warp

# Turn the PyVista mesh into a PolyMesh
mesh = PolyMesh.from_vtk(ugrid)
mesh.default_color = 'gray'

Scene([mesh])
```