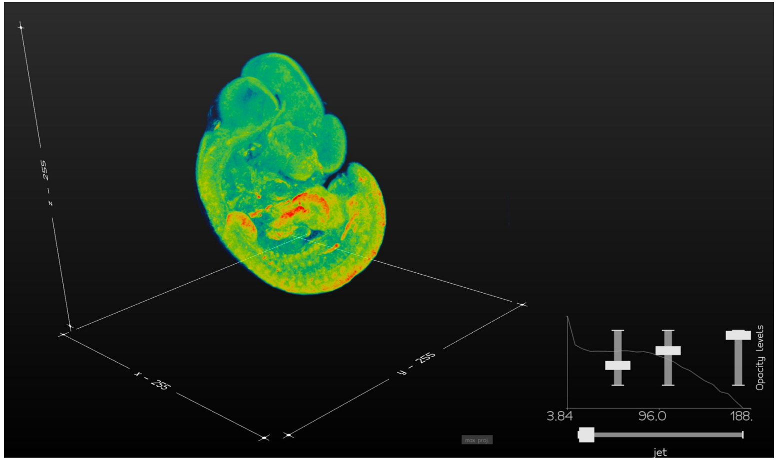
vədo

A python module for scientific **analysis** and **v**isualization of **əd o**bjects



Marco Musy, Johannes Soltwedel

EMBL, Physics of Life, BIAS Course - August 2023

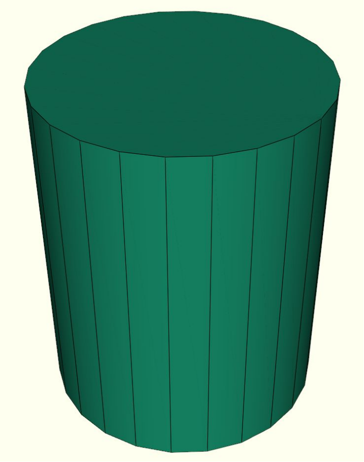
Why?

`matplotlib` is not very useful in 3D

VTK is a fantastic library… but it has a steep learning curve.

```
import vtk


def main():
    colors = vtk.vtkNamedColors()
    # Set the background color.
    bkg = map(lambda x: x / 255.0, [26, 51, 102, 255])
    colors.SetColor("BkgColor", *bkg)

    # This creates a polygonal cylinder model with eight circumferential
    # facets.
    cylinder = vtk.vtkCylinderSource()
    cylinder.SetResolution(8)

    # The mapper is responsible for pushing the geometry into the graphics
    # library. It may also do color mapping, if scalars or other
    # attributes are defined.
    cylinderMapper = vtk.vtkPolyDataMapper()
    cylinderMapper.SetInputConnection(cylinder.GetOutputPort())

    # The actor is a grouping mechanism: besides the geometry (mapper), it
    # also has a property, transformation matrix, and/or texture map.
    # Here we set its color and rotate it -22.5 degrees.
    cylinderActor = vtk.vtkActor()
    cylinderActor.SetMapper(cylinderMapper)
    cylinderActor.GetProperty().SetColor(colors.GetColor3d("Tomato"))
    cylinderActor.RotateX(30.0)
    cylinderActor.RotateY(-45.0)

    # Create the graphics structure. The renderer renders into the render
    # window. The render window interactor captures mouse events and will
    # perform appropriate camera or actor manipulation depending on the
    # nature of the events.
    ren = vtk.vtkRenderer()
    renWin = vtk.vtkRenderWindow()
    renWin.AddRenderer(ren)
    iren = vtk.vtkRenderWindowInteractor()
    iren.SetRenderWindow(renWin)

    # Add the actors to the renderer, set the background and size
    ren.AddActor(cylinderActor)
    ren.SetBackground(colors.GetColor3d("BkgColor"))
    renWin.SetSize(300, 300)
    renWin.SetWindowName('CylinderExample')

    # This allows the interactor to initalize itself. It has to be
    # called before an event loop.
    iren.Initialize()

    # We'll zoom in a little by accessing the camera and invoking a "Zoom"
    # method on it.
    ren.ResetCamera()
    ren.GetActiveCamera().Zoom(1.5)
    renWin.Render()

    # Start the event loop.
    iren.Start()


if __name__ == '__main__':
    main()
```

```
import vedo
vedo.Cylinder().show()
```



…not only visualization!

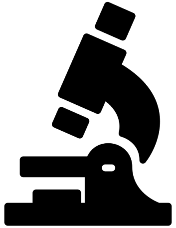
(paraview can already do it)

Vedo makes working with VTK a lot easier. I do understand VTK (or at least I think I do), but it is still a lot of work to get something simple done!

R. de Bruin, Delft Univ. of Tech

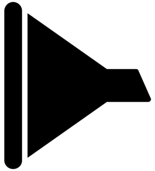
Where?

vedo sits somewhere in here



TB

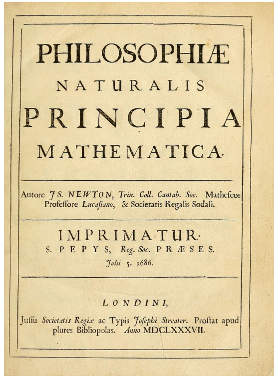
Preprocessing
Raw Data



GB

Postprocessing
Data analysis

MB


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore J S. NEWTON, Trin. Coll. Cantab. Soc. Matheseos Professore Lucasiano, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, Reg. Soc. PRÆSES.
Julii 5. 1686.

LONDINI,
Jussu Societatis Regiæ ac Typis Josephi Streater. Prostat apud plures Bibliopolas. Anno MDCLXXXVII.

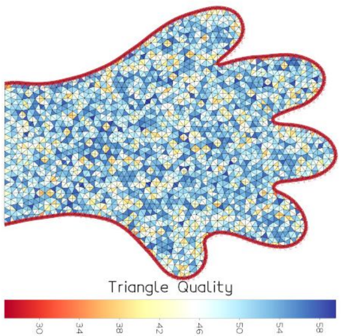
“A 3D-powered version of `matplotlib`”

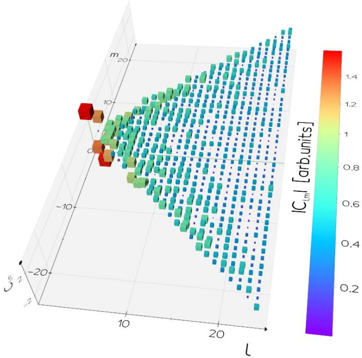
“A handy day-to-day tool for the researcher”

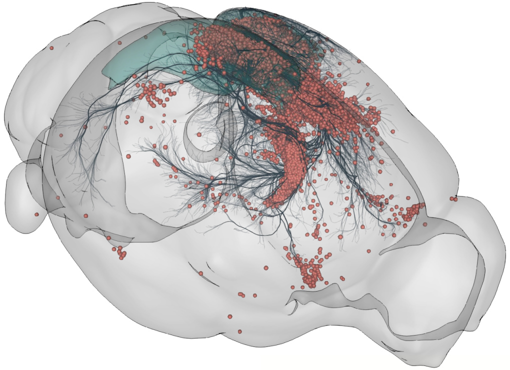
It can prove useful with any type of data having a spatio-temporal structure

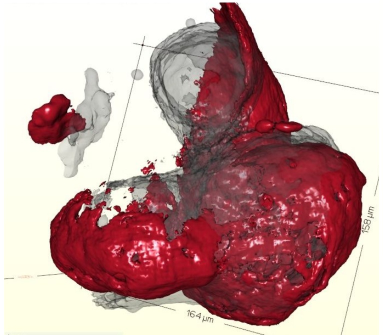
What can *you* do with it?

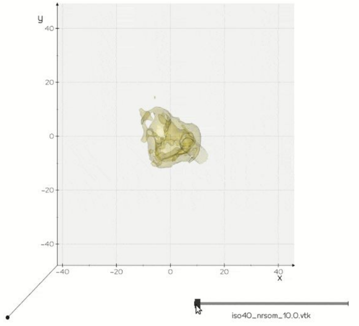
- Work with polygonal meshes and point clouds
- Morphometrics (mesh warp, cut, connect, ...)
- Analysis of 3D images and tetrahedral meshes
- 2D/3D plotting and histogramming.
- Integration with other external libraries (`Qt`, `napari`, `trimesh`, `pymeshlab`, `SHTools` ...)
- Jupyter and Colab environments are supported
- `Command Line Interface` (CLI) as quick viz tool
- Export/exchange 3D interactive scenes to file
- Create interactive animations
- Generate publication-quality renderings



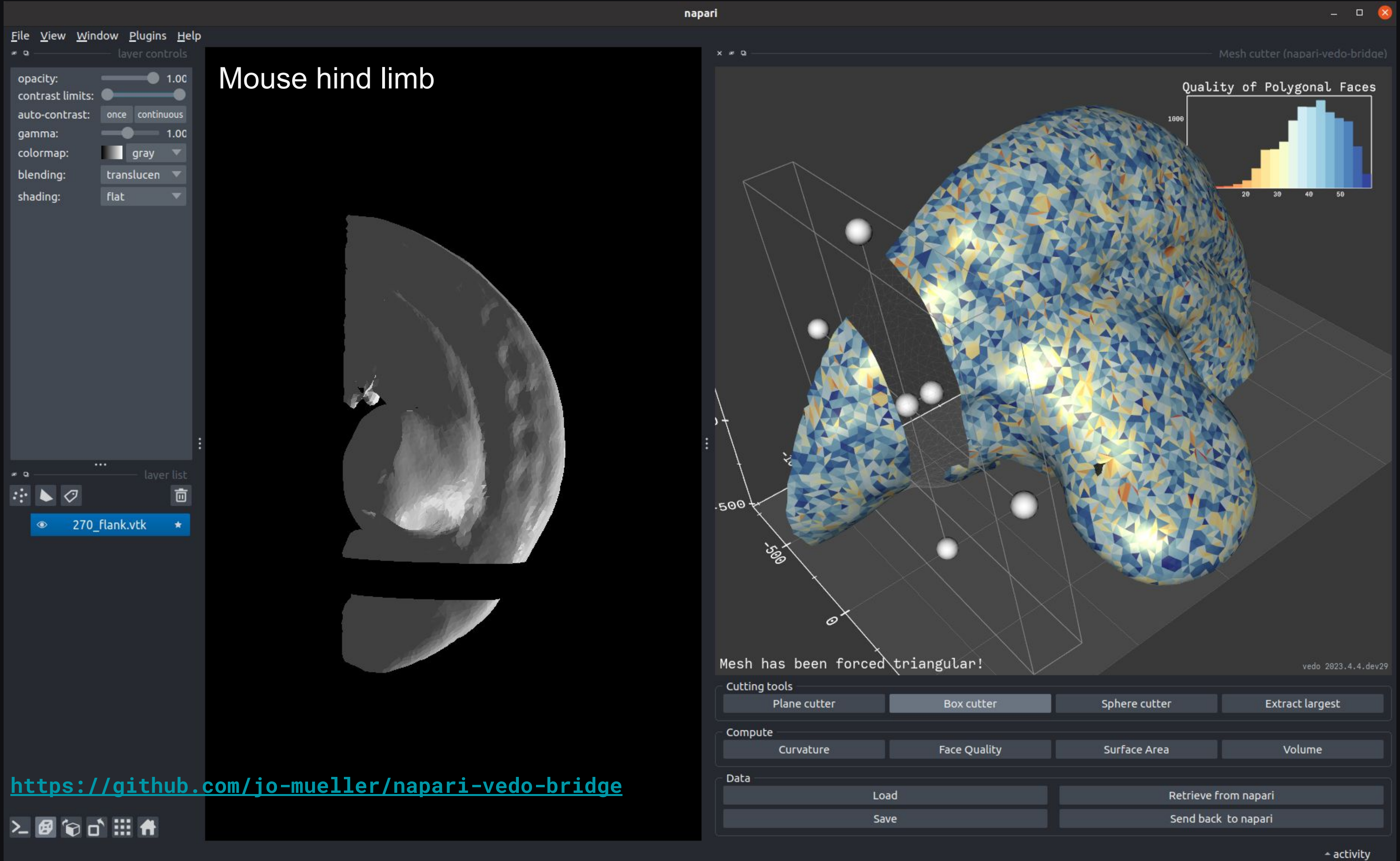








Mouse hind limb



https://github.com/jo-mueller/napari-vedo-bridge

How?

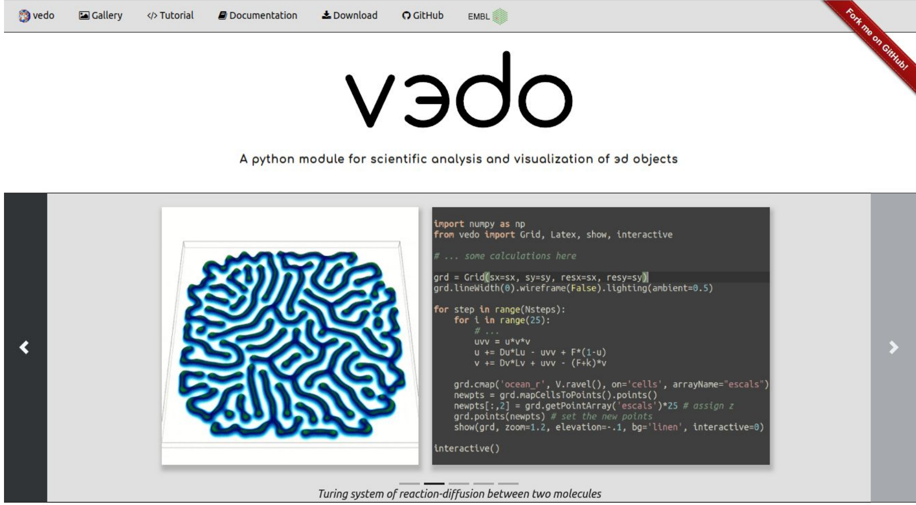
- Install:

  `pip install vedo`

- Documentation:

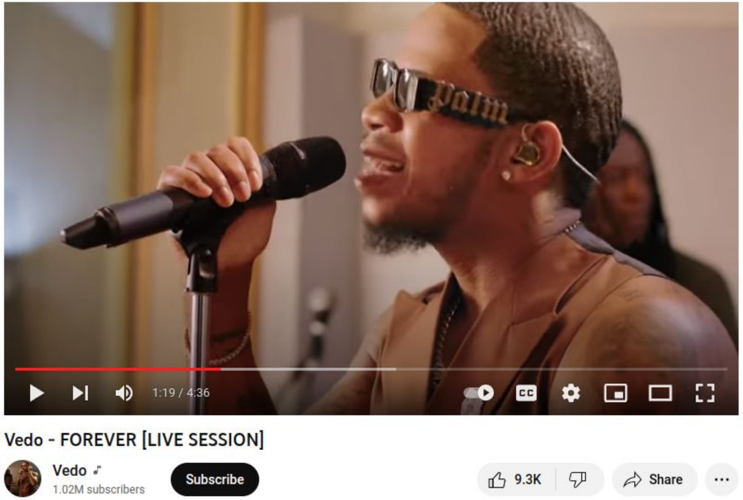
  https://vedo.embl.es/

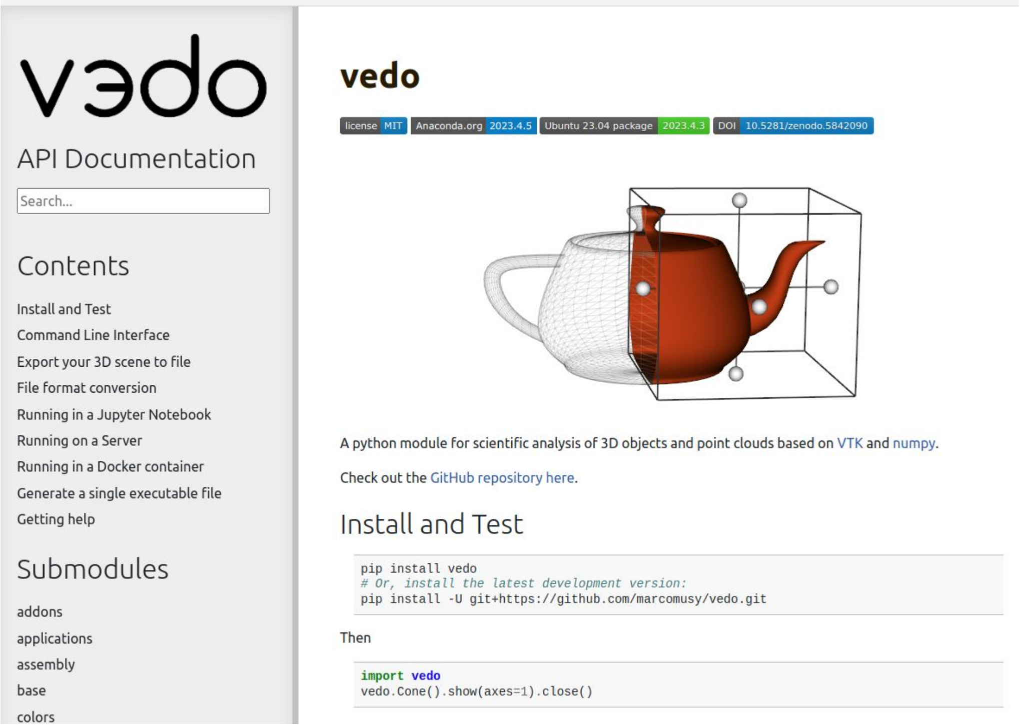
- **350+** examples as reference
  - Designed to be short and intuitive (most are <30 lines)
  - **Searchable** `vedo --search string`
  - **Runnable** `vedo --run examplename`



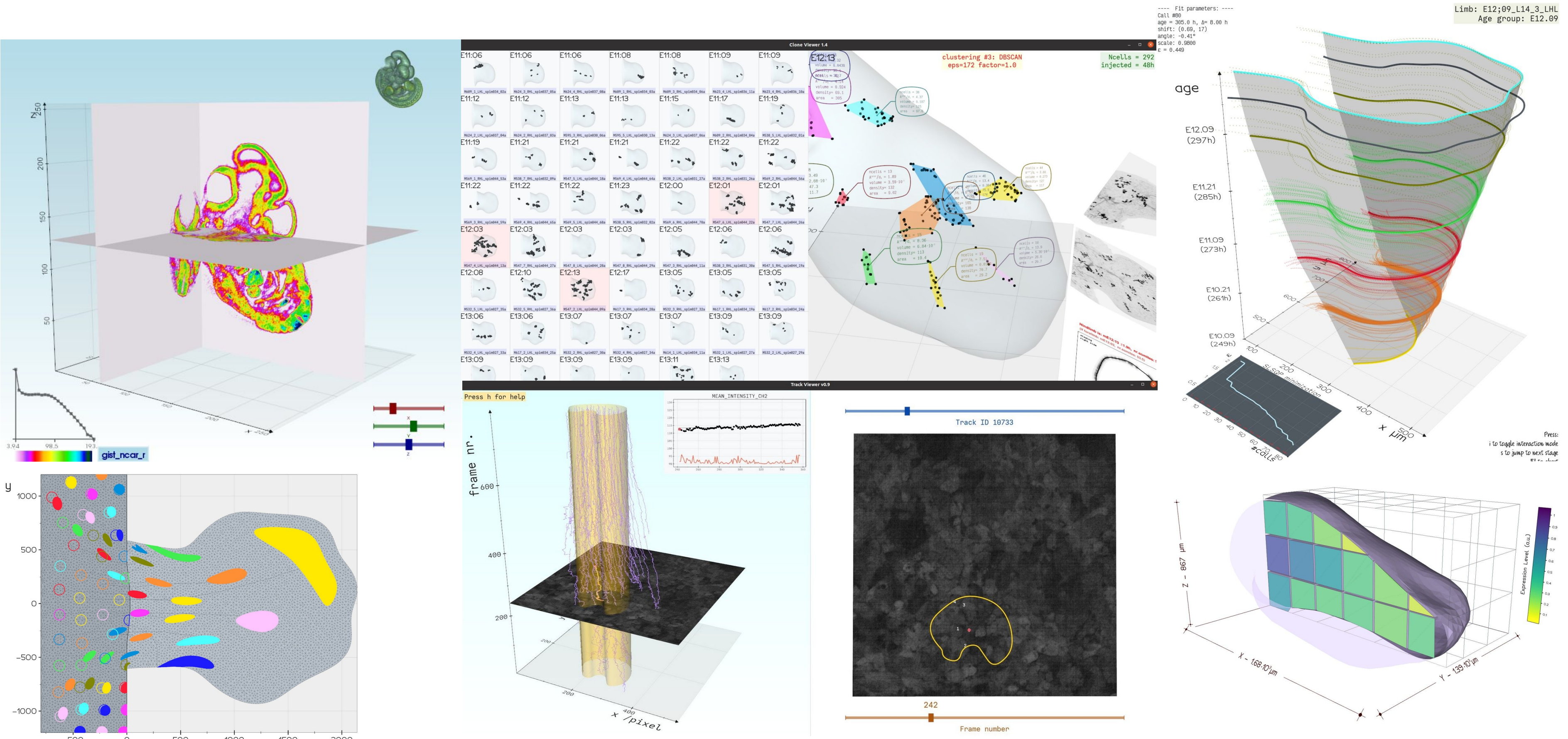
Turing system of reaction-diffusion between two molecules

API documentation is found at vedo.embl.es/docs





Play with your own data (if you have it at hand!)



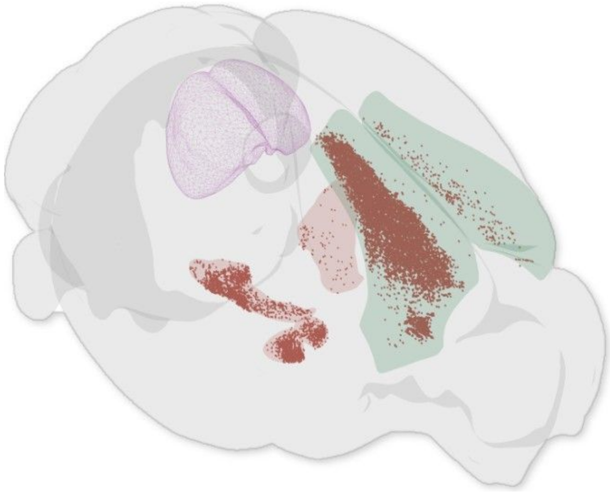
Conclusion

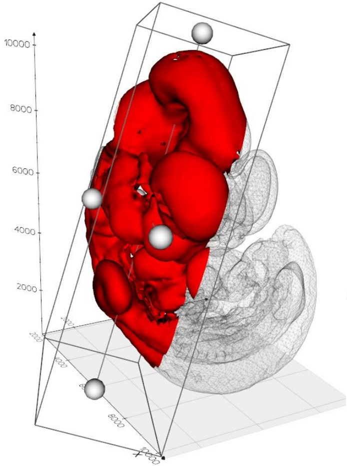
- Proved very useful in diverse applications
- Documented API with many examples
- **Happy to offer support!**

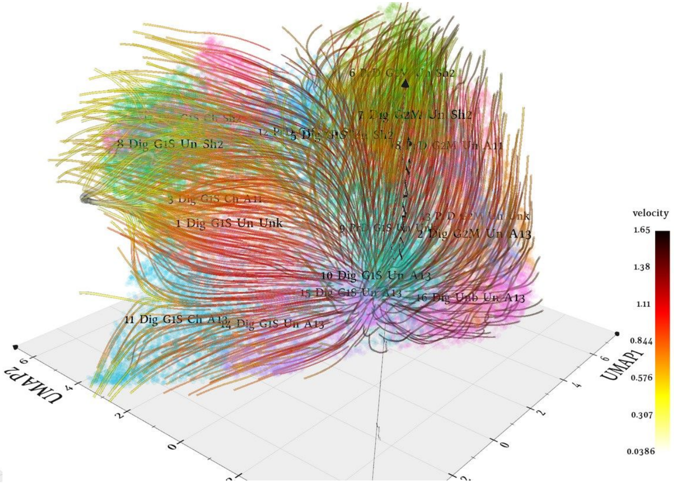
marco.musy@embl.es

johannes_richard.soltwedel@tu-dresden.de

https://vedo.embl.es/

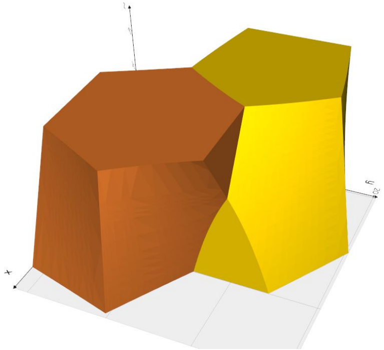






Pol BIAS, August 2023

vedo practicals



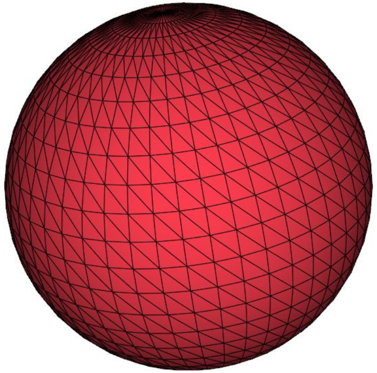
```
> pip install vedo
```

01-basics.ipynb

Basic Geometric Objects

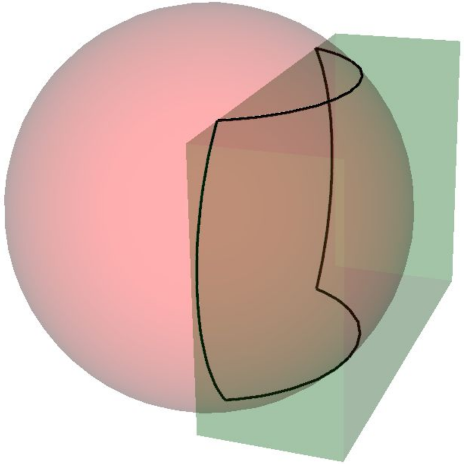
```python
from vedo import *

settings.default_backend = "vtk"

sphere = Sphere().linewidth(1)

plt = Plotter()
plt += sphere
plt.show()
plt.close()
```

In Jupyter notebooks



Press “h” in rendering window

```python
# Create a sphere and a box
sphere = Sphere(r=1.5).c("red5", 0.2)
box = Box(pos=(1,0,0)).triangulate().c("green5", 0.2)

# Find the intersection between the two
intersection = sphere.intersect_with(box).lw(4)

plt += [sphere, box, intersection]
```

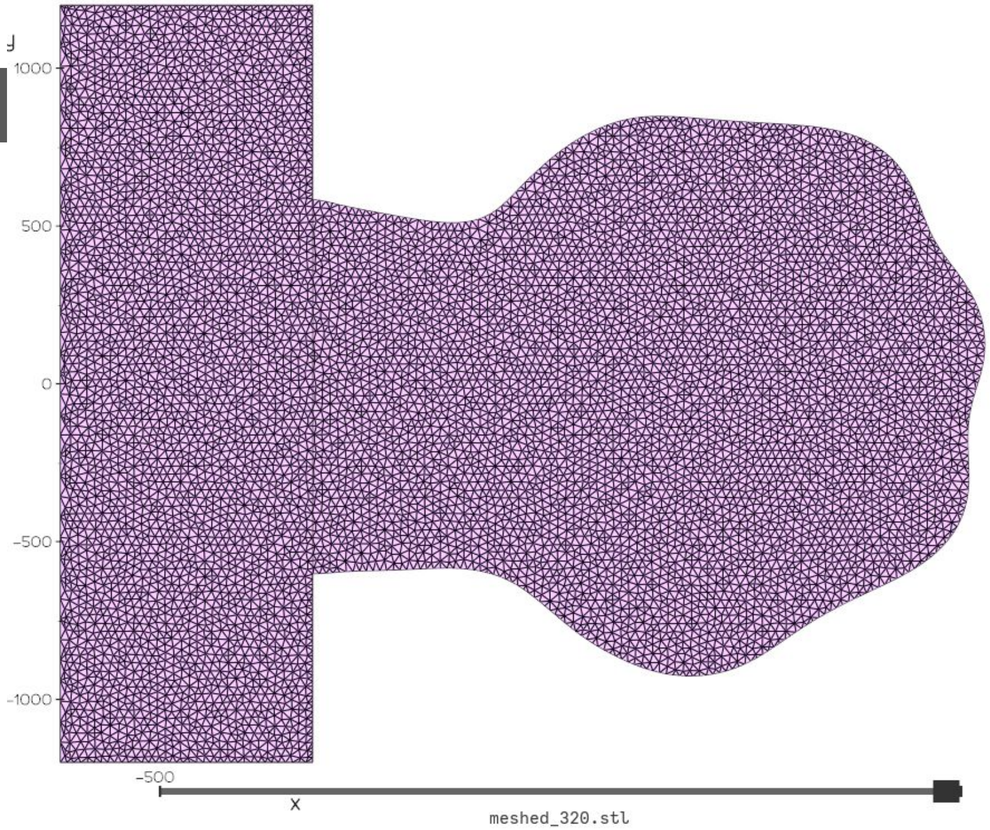


Plotting made simple

```
> vedo -s data/meshed_*.stl
```

Or in a script:

```python
from vedo import load
from vedo.applications import Browser

meshes = load("../data/meshed_*.stl")

for i in range(len(meshes)):
    meshes[i].color(i).linewidth(1)

bro = Browser(meshes, size=(1600,800))
bro.show()
bro.close()
```

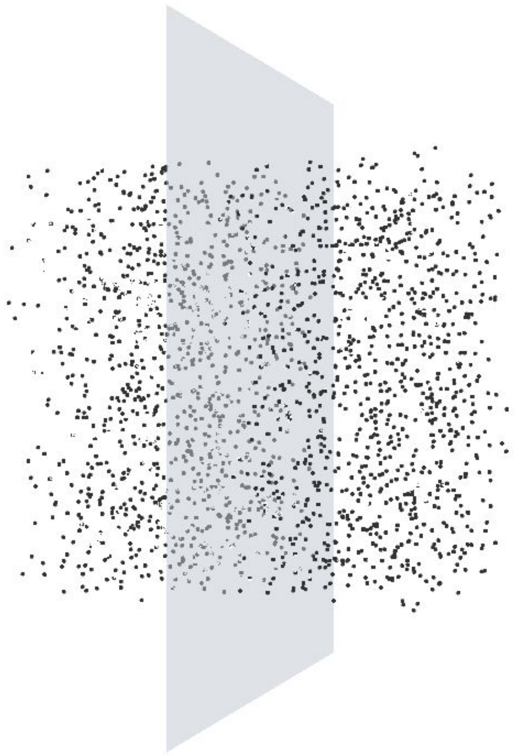


`scripts/01-vis_serie.py`

Point Clouds & Polygonal Meshes

Create a point cloud and cut it

```
points = np.random.rand(2000, 3)

pts = Points(points)
pln = Plane(pos=(0.5, 0.5, 0.6), normal=(1, 0, 0), s=(1.5, 1.5))

show(pts, pln).close()
```



Create a point cloud and cut it

```
pts.cut_with_plane((0.5, 0.5, 0.6))
```

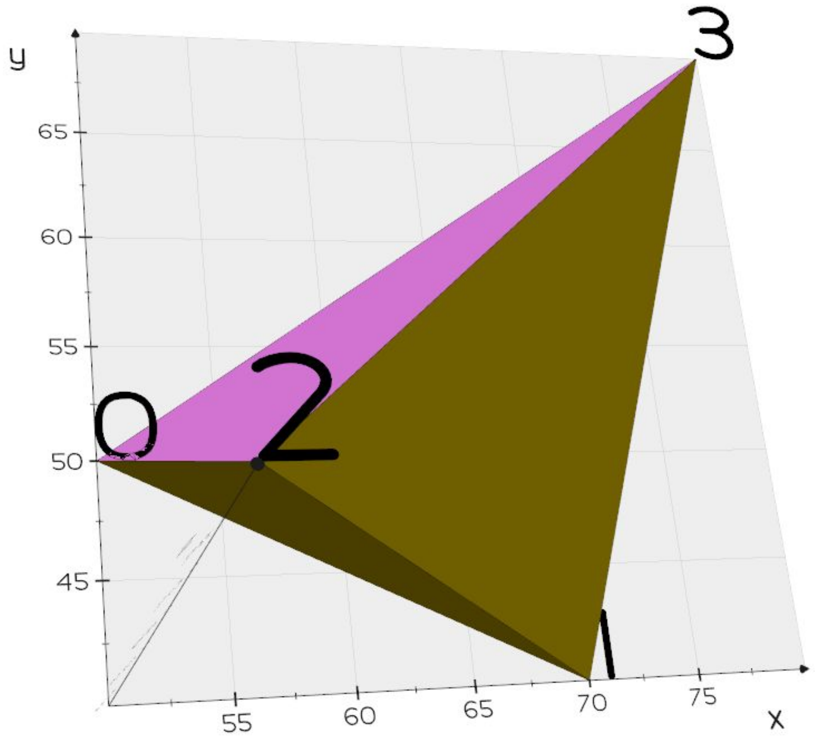
🔥 Can you create a normal distributed cloud and cut the points inside a cylinder?

Hint: search the [API docs]() for "cylinder"

Build a polygonal mesh manually

```python
from vedo import Mesh, show

# Define the vertices and faces that make up the mesh
verts = [(50,50,50), (70,40,50), (50,40,80), (80,70,50)]
faces = [(0,1,2), (2,1,3), (1,0,3)]

# Build the polygonal Mesh object from the vertices and faces
mesh = Mesh([verts, faces])

# Set the backcolor of the mesh to violet
# and show edges with a linewidth of 2
mesh.backcolor('violet').linecolor('tomato').linewidth(2)

# Create labels for all vertices in the mesh showing their ID
labs = mesh.labels('id').c('black')

# Print the points and faces of the mesh as numpy arrays
print('points():', mesh.points())
print('faces() :', mesh.faces())

# Show the mesh, vertex labels, and docstring
show(mesh, labs, viewup='z', axes=1).close()
```

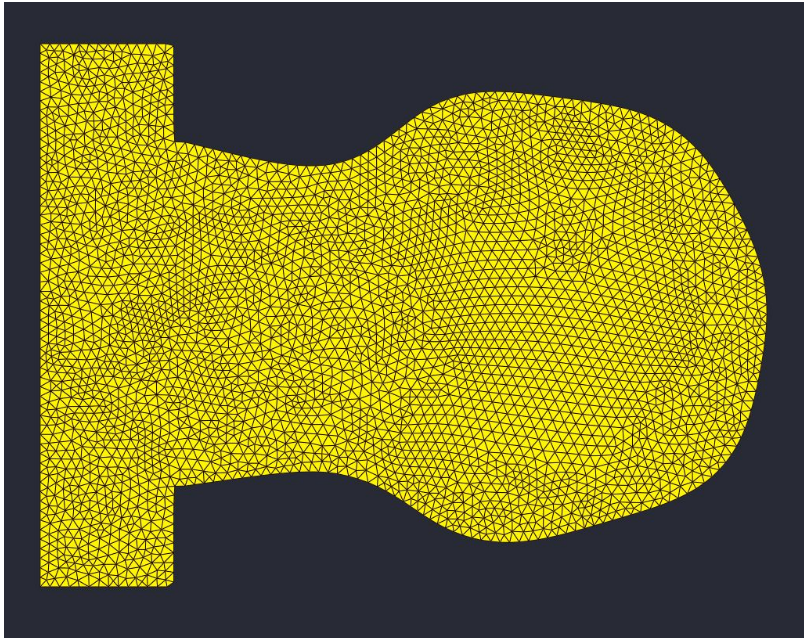
`vedo --run buildmesh`



Create a polygonal mesh

```
# Read data
faces = np.load(faces_path)
verts = np.load(verts_path)
```

```
msh = Mesh([verts, faces]).linewidth(1)
```



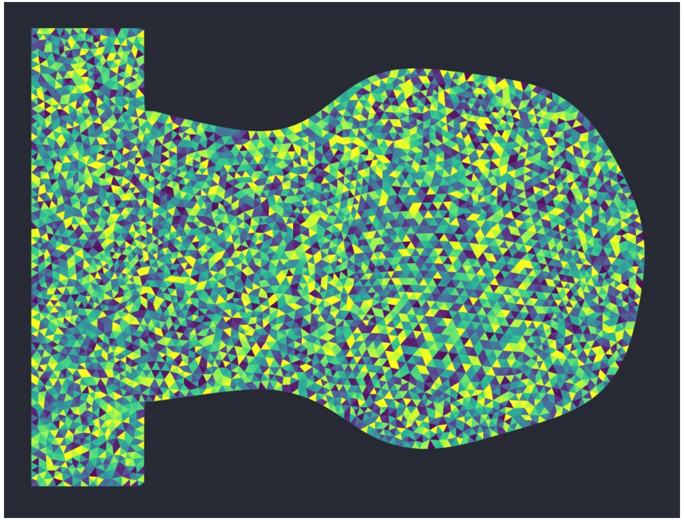
05-gene_mesh.ipynb

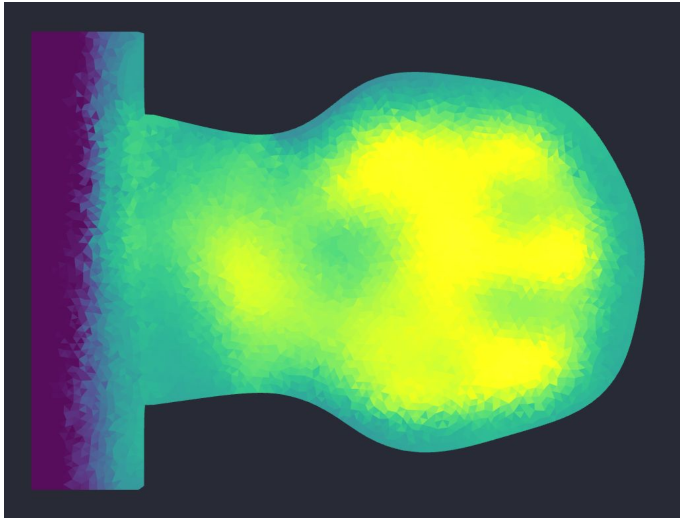
Add gene data associated to cells

```
# Adding scalar values
n = faces.shape[0] # number of faces
values = np.random.random(n)
msh.celldata["fake_data"] = values
```

🔥 Can you add the gene expression data to the mesh?

Use ../data/gene_data.npy

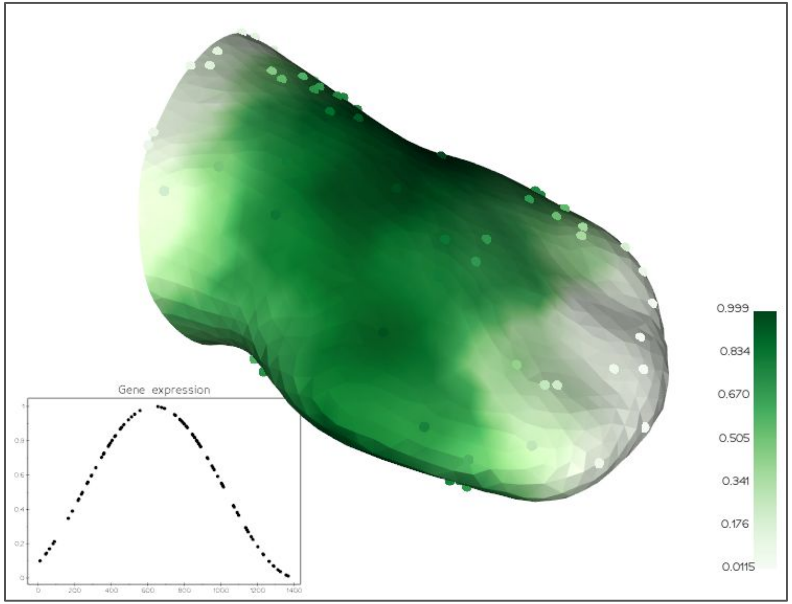




Interpolate data from sparse measurements in 3D

Let’s assume that we know the expression of a gene in 100 positions

```python
import numpy as np
from vedo import dataurl, Mesh, Points, show
from vedo.pyplot import plot

# Load a mesh of a mouse limb at 12 days of development
msh = Mesh(dataurl + "290.vtk")

# Pick 100 points where we measure the value of a gene expression
ids = np.random.randint(0, msh.npoints, 100)
pts = msh.points()[ids]        # slice the numpy array
x = pts[:, 0]                  # x coordinates of the points
gene = np.sin((x+150)/500)**2  # we are making this up!

# Create a set of points with those values
points = Points(pts, r=10).cmap("Greens", gene)

# Interpolate the gene data onto the mesh, by averaging the 5 closest points
msh.interpolate_data_from(points, n=5).cmap("Greens").add_scalarbar()

# Create a graph of the gene expression as function of x-position
gene_plot = plot(x, gene, lw=0, title="Gene expression").as2d(scale=0.5)

# Show the mesh, the points and the graph
show(msh, points, gene_plot)
```

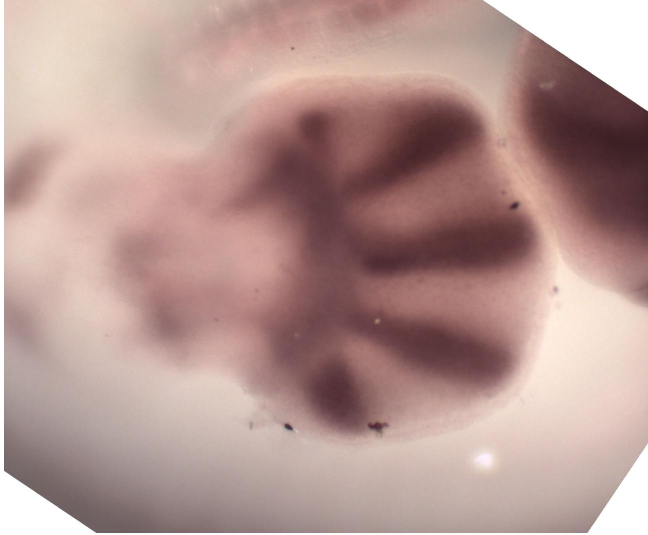


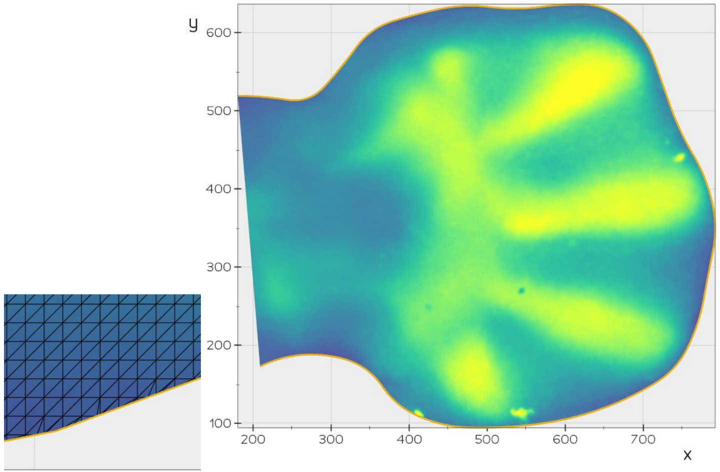
07-grab_scalars.ipynb

Mesh from a JPG image

Manually select a contour and extract a **polygonal mesh** from the input image

```
from vedo import Picture, settings, show
from vedo.applications import SplinePlotter

settings.default_backend = "vtk"

pic = Picture("data/sox9_exp.jpg").bw()

plt = SplinePlotter(pic)
plt.show(mode="image", zoom='tight')
outline = plt.line
plt.close()

print("Cutting with outline...")
msh = pic.tomesh().triangulate().cmap("viridis_r")
cut_msh = msh.clone().cut_with_point_loop(outline)

cut_msh.interpolate_data_from(msh, n=3)
show(cut_msh, outline, axes=1).close()
```



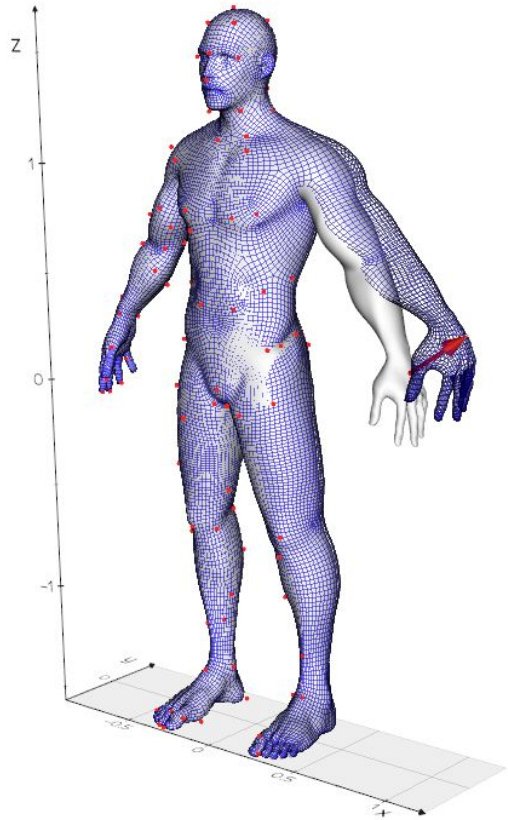


08-warp_mesh.ipynb

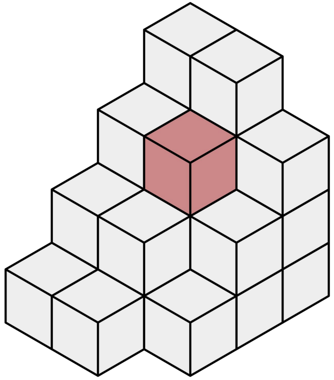
Warp a Mesh (non-linear registration)

All points stay fixed while a single point in space moves as the arrow indicates

```python
from vedo import dataurl, Mesh, Arrows, show

# Load a mesh
mesh = Mesh(dataurl+"man.vtk").color("white")

# Create a heavily decimated copy with about 200 points
# (to speed up the computation)
mesh_dec = mesh.clone().triangulate().decimate(n=200)

sources = [[0.9, 0.0, 0.2]]  # this point moves
targets = [[1.2, 0.0, 0.4]]  # ...to this.
for pt in mesh_dec.points():
    if pt[0] < 0.3:          # while these pts don't move
        sources.append(pt)   # (e.i. source = target)
        targets.append(pt)

# Create the arrows representing the displacement
arrow = Arrows(sources, targets)

# Warp the mesh
mesh_warped = mesh.clone().warp(sources, targets)
mesh_warped.c("blue").wireframe()

# Show the meshes and the arrow
show(mesh, mesh_warped, arrow, axes=1)
```



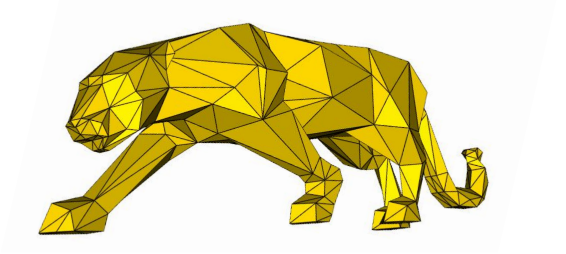
Volumes
(eg TIFF stacks)

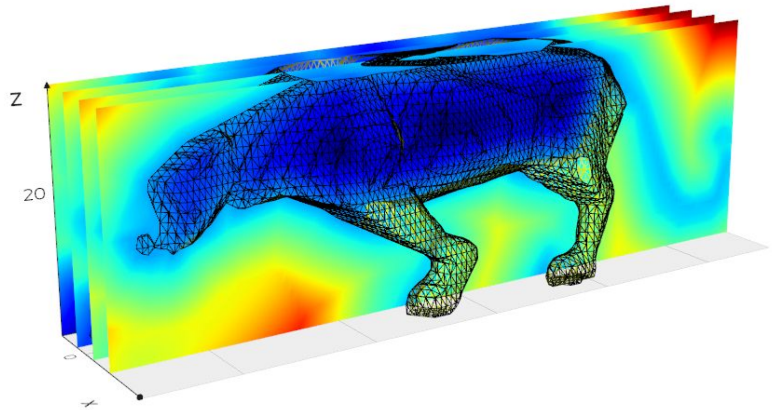


Compute distance from a mesh

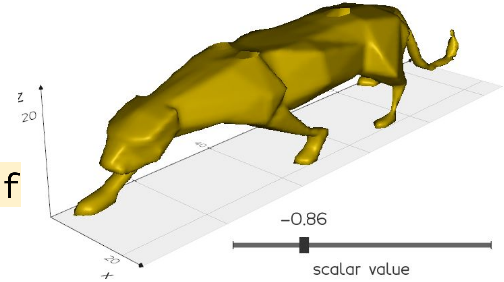
..and save it as a tiff stack

```python
from vedo import *

msh = Mesh(dataurl + "panther.stl")

vol = msh.signed_distance(dims=[25,125,25])
iso = vol.isosurface(0.0)

plt = Plotter()
plt += iso.wireframe()

for i in range(0, 25, 5):
    plt += vol.xslice(i).cmap("jet")

vol.write("panther.tif")
plt.show(axes=1)
```






```
> vedo panther.tif
```

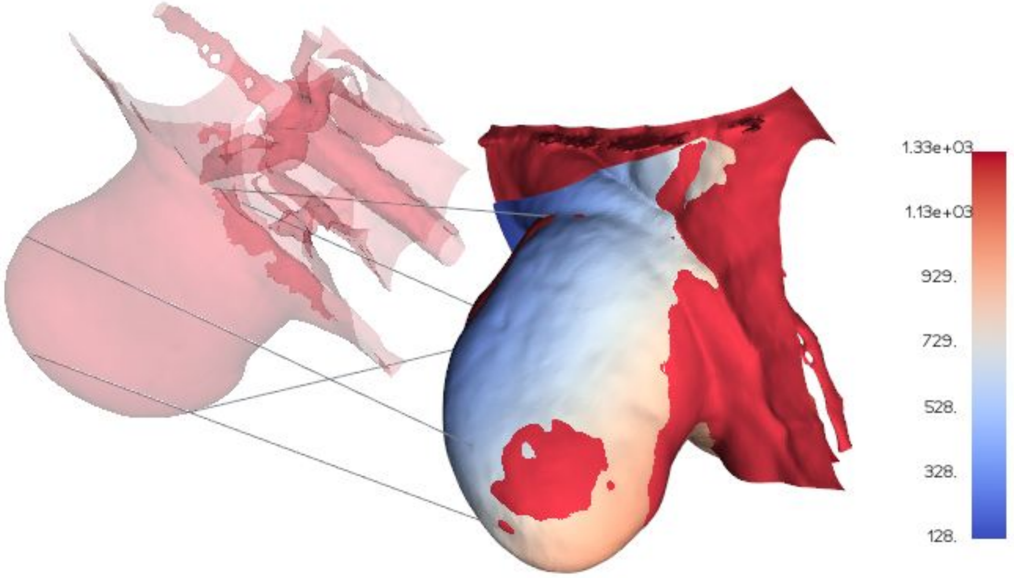


Extras: Linear vs Non-Linear Registration

Naive registration (ICP) might fail!

Try to:

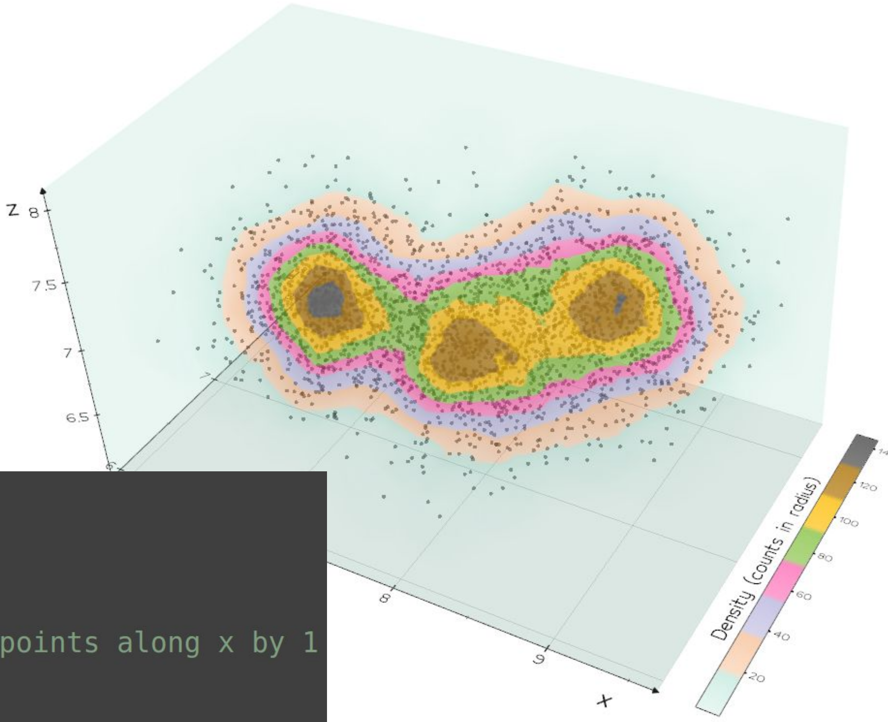
1. Inspect the meshes
2. Identify landmark points
3. Warp one mesh onto the other



Compute point density

..from a point cloud:

`14-cells_segment.ipynb`



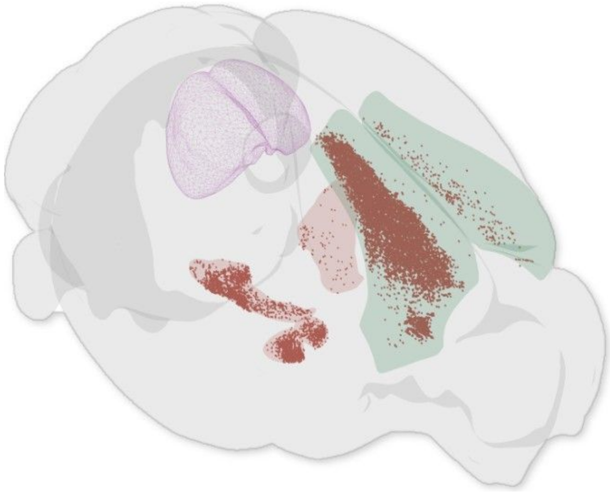

```python
from vedo import *

n = 3000
p = np.random.normal(7, 0.3, (n,3))
p[:int(n*1/3) ] += [1,0,0]       # shift 1/3 of the points along x by 1
p[ int(n*2/3):] += [1.7,0.4,0.2]

pts = Points(p, alpha=0.5)

# Create a Volume from the points representing the points density
vol = pts.density(radius=0.25).cmap('Dark2').alpha([0.1,1])
vol.add_scalarbar3d(title='Density (counts in radius)', c='k')

show(pts, vol, __doc__, axes=1).close()
```

Conclusion

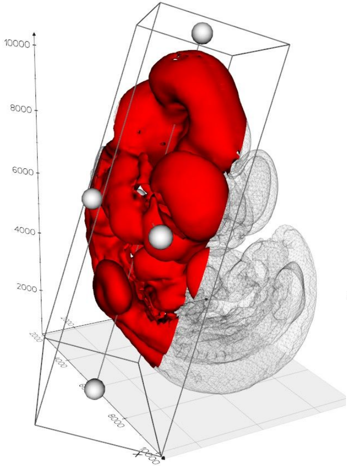
- Proved very useful in diverse applications
- Documented API with many examples
- **Happy to offer support!**

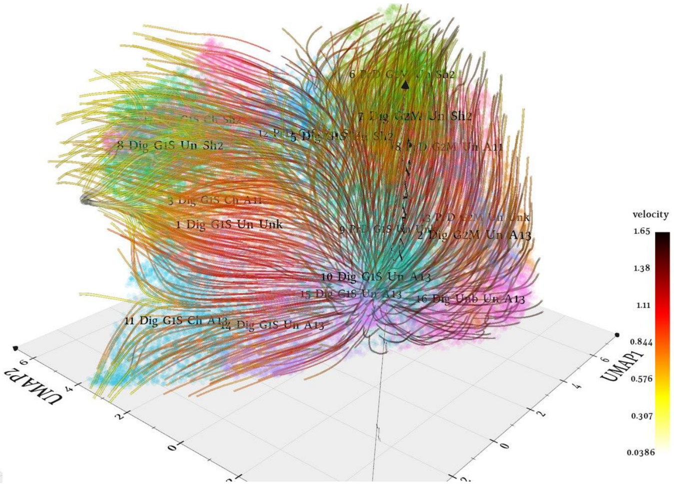
marco.musy@embl.es

johannes_richard.soltwedel@tu-dresden.de



https://vedo.embl.es/





Pol BIAS, August 2023