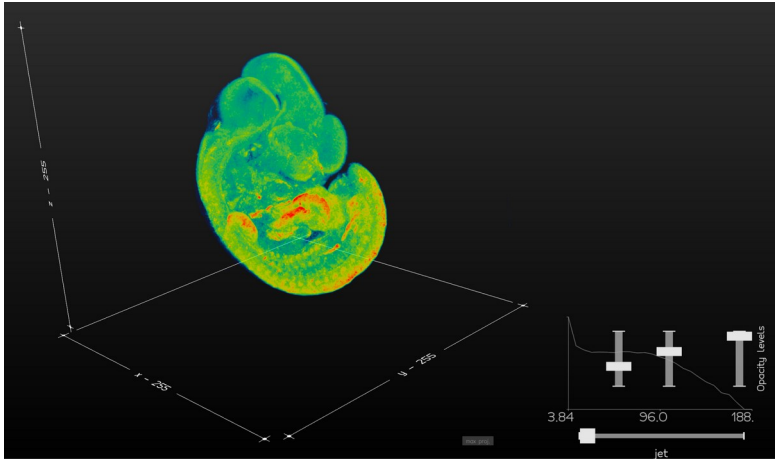
EMBL European Molecular Biology Laboratory

Sharpe Lab

GitHub

vedo

A python module for scientific analysis and visualization of 3d objects



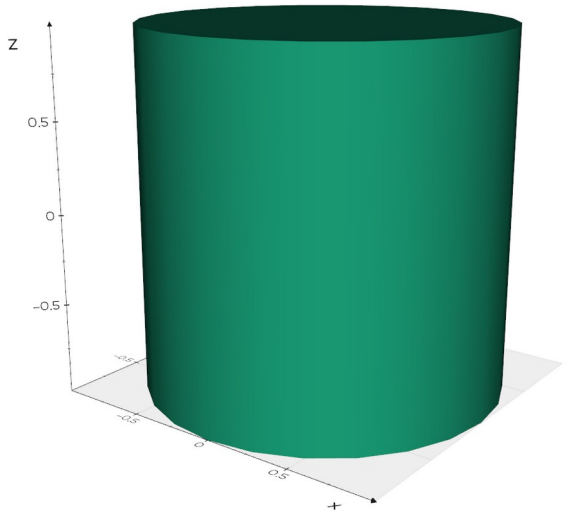
Marco Musy

EMBL, EPUG Seminar - 12th December 2023

Why do we need this?

```
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()
```

"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.




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

Where?

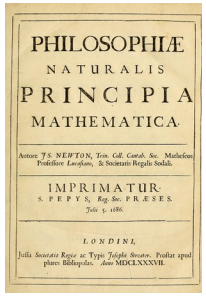
vedo sits somewhere in here

TB

Preprocessing
Raw Data

GB

Postprocessing
Data analysis



MB

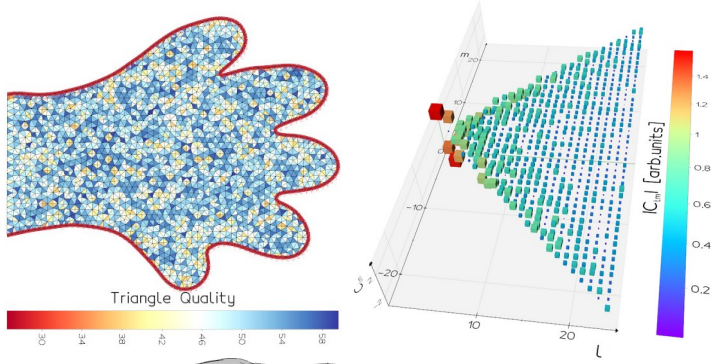
"A 3D-powered version of `matplotlib`"

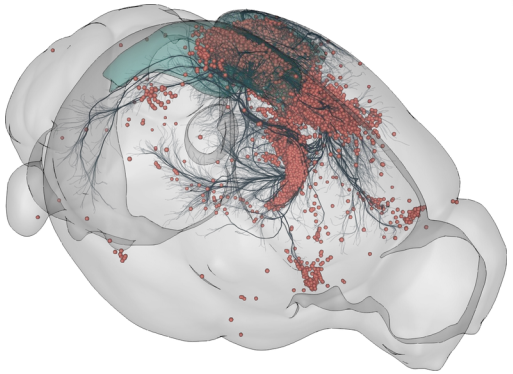
"A handy day-to-day tool for the researcher"

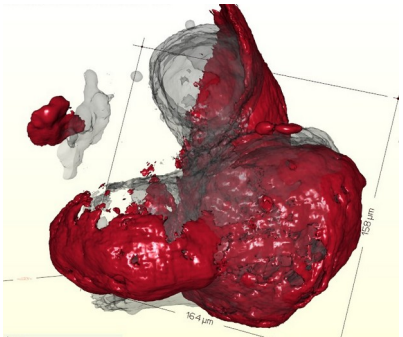
It can prove useful with any type of data having a spatial-temporal structure

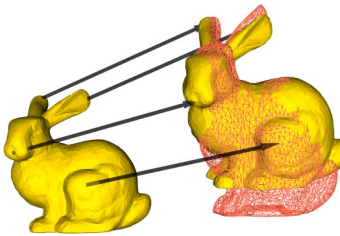
What can *you* do with it?

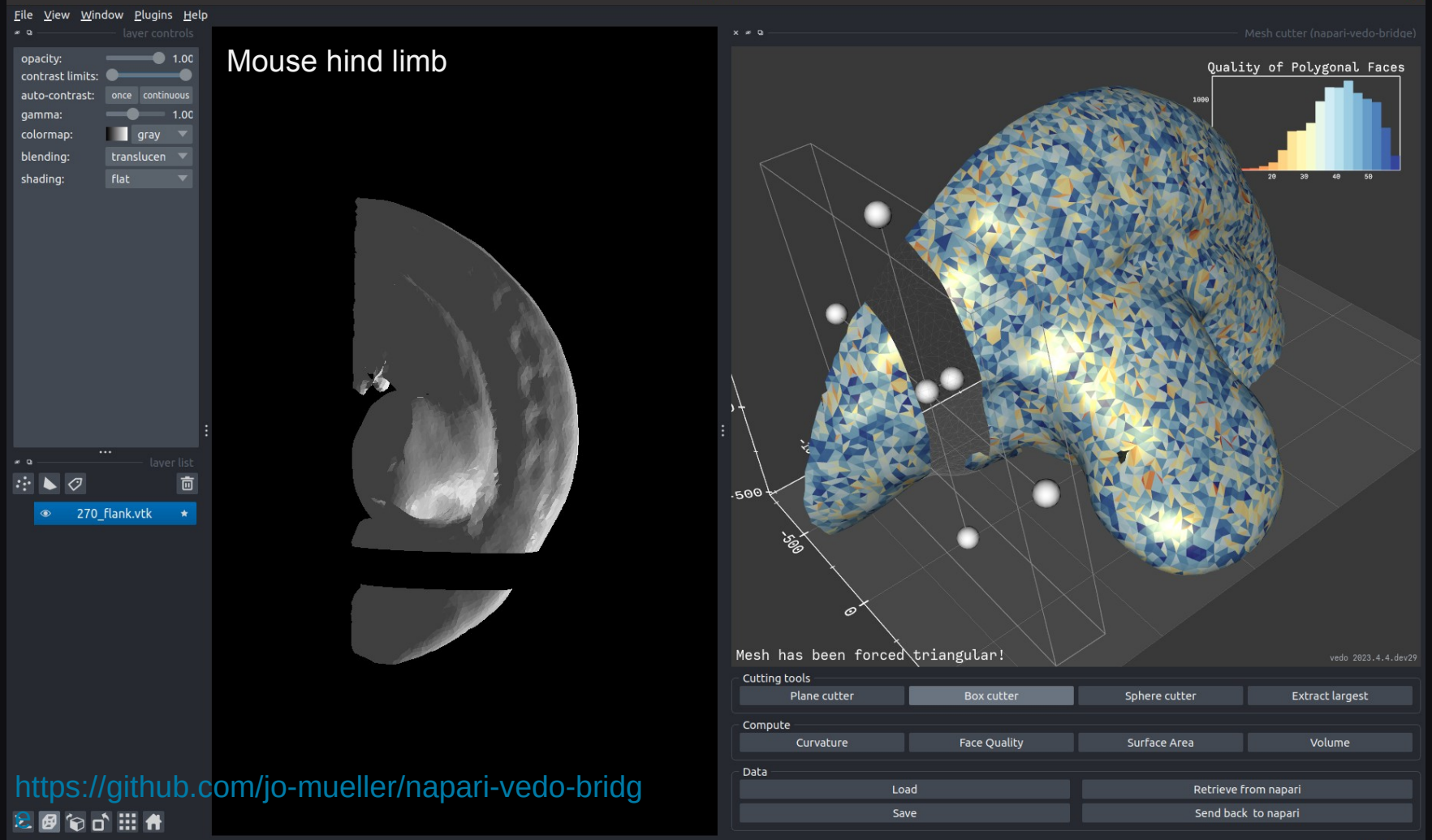
- Work with polygonal meshes and point clouds
- Morphometrics (mesh warp, cut, connect, …)
- Analysis of 3D images and tetrahedral meshes
- 2D/3D plotting and histogramming.
- Integration with other external libraries (`Qt`, `napari`, `trimesh`, `meshlab`, `SHTools` …)
- Jupyter and Colab environments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










napari
File View Window Plugins Help
layer controls
opacity: 1.00
contrast limits:
auto-contrast: once continuous
gamma: 1.00
colormap: gray
blending: translucen
shading: flat
layer list
270_flank.vtk
Mouse hind limb
Mesh cutter (napari-vedo-bridge)
Quality of Polygonal Faces
1000
20 30 40 50
-500
-500
0
Mesh has been forced triangular!
vedo 2023.4.4.dev29
Cutting tools
Plane cutter
Box cutter
Sphere cutter
Extract largest
Compute
Curvature
Face Quality
Surface Area
Volume
Data
Load
Retrieve from napari
Save
Send back to napari
activity
https://github.com/jo-mueller/napari-vedo-bridg

How?

- Install:

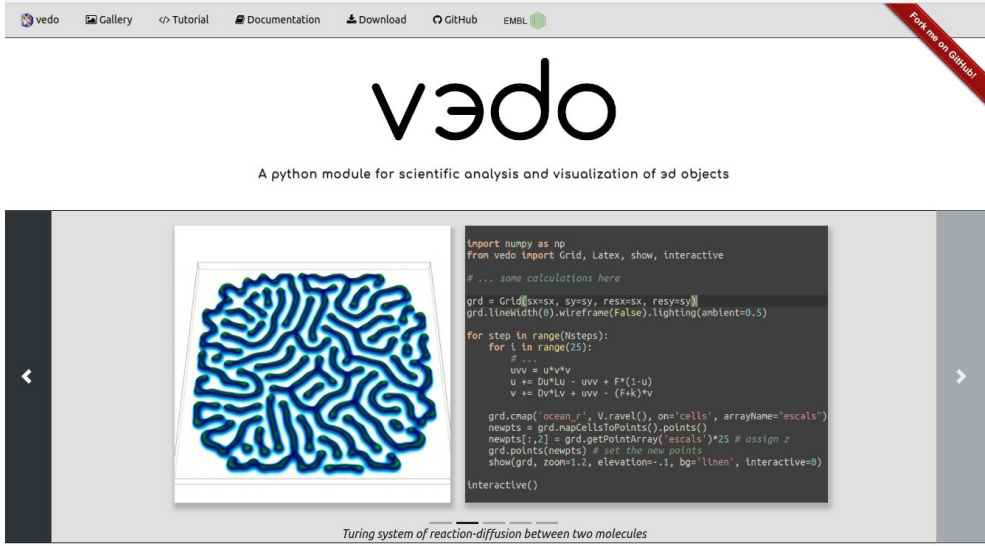
  `pip install vedo`

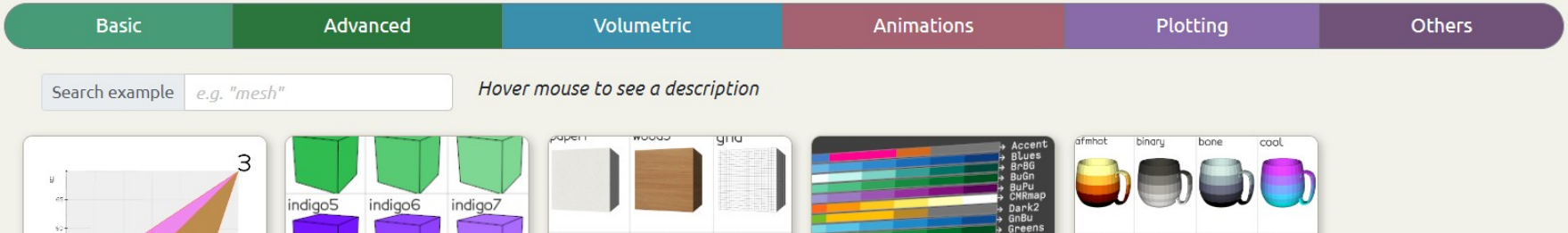
- Web page:

  https://vedo.embl.es/

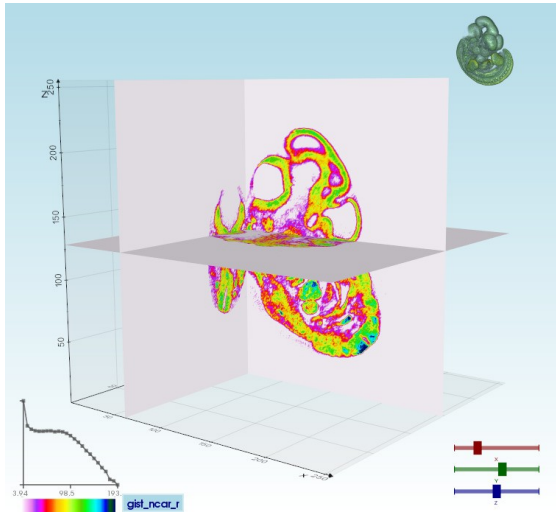
- **350+** examples as reference
  - Designed to be short and intuitive (most are <30 lines)
  - Searchable `vedo --search string`
  - Runnable `vedo --run examplename`

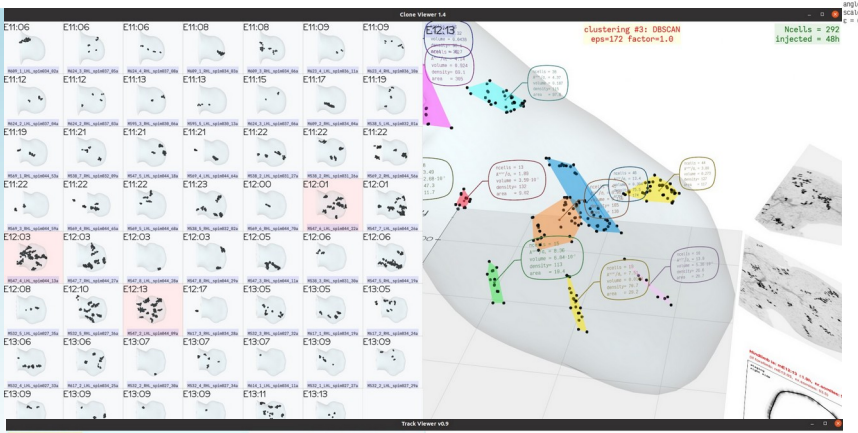
  API documentation available at vedo.embl.es/docs

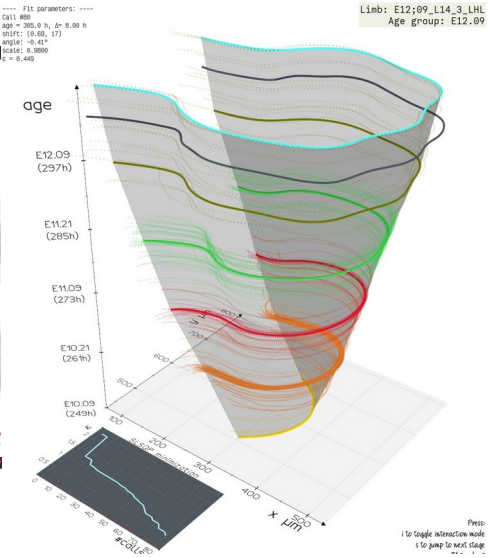


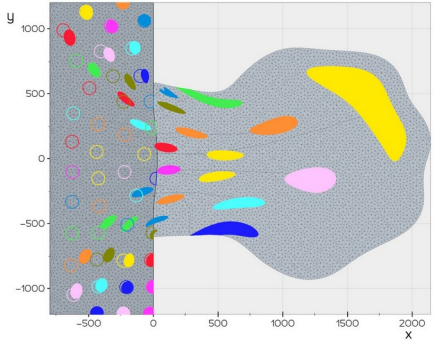


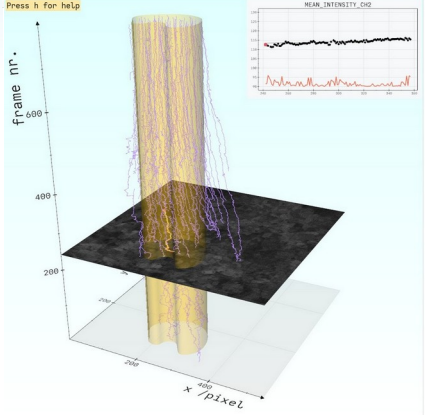
Play with your own data (if you have it at hand!)

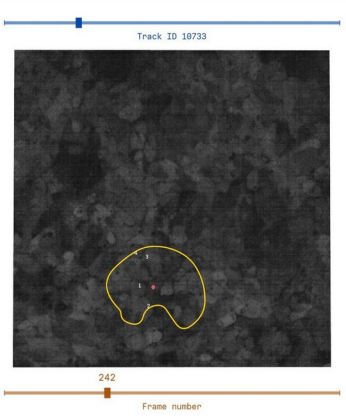


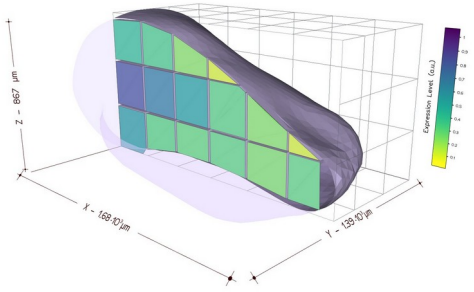








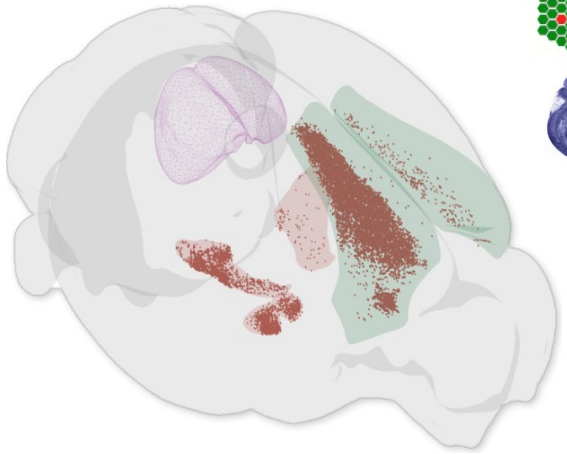




Conclusion

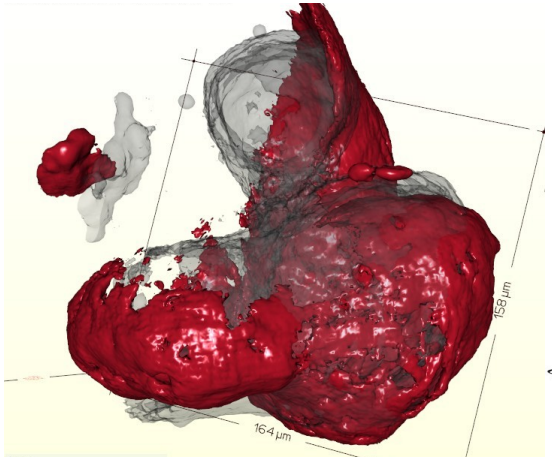
- Proved very useful in diverse applications
- Documented API with many examples
- **Happy to offer support!**

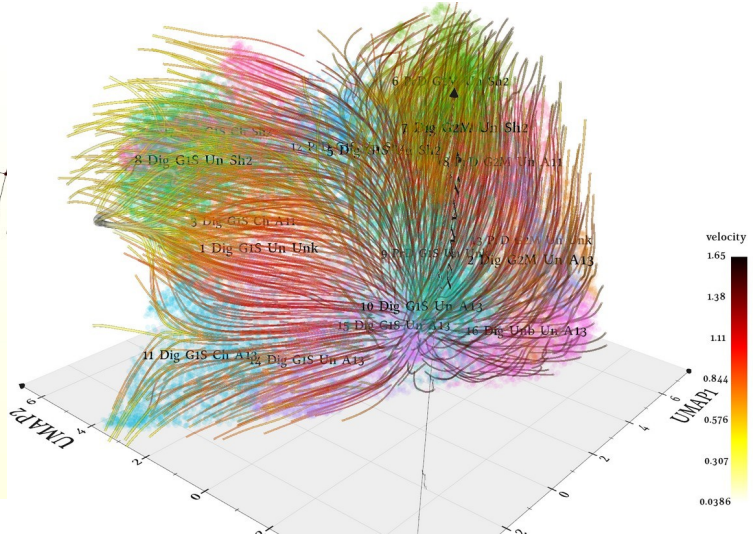
marco.musy@embl.es



https://vedo.embl.es/

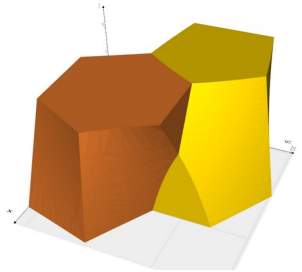
"I really enjoyed using [vedo] for making fairly complex models",
A. Pollack – SCRF, Stanford





EMBL, EPUG Dec. 2023

practicals/gallery



```
pip install vedo -U

pip install scipy

git clone https://github.com/vedo-epug-tutorial.git
```

LINK

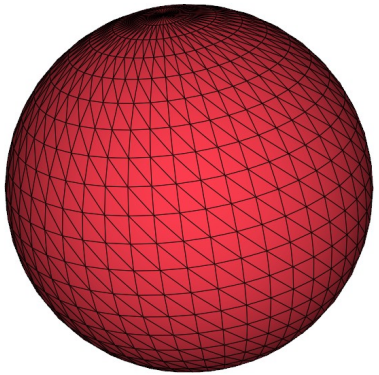
```
cd vedo-epug-tutorial
```

Note: this is only a relatively small subset of what is possible!

Basic Geometric Objects

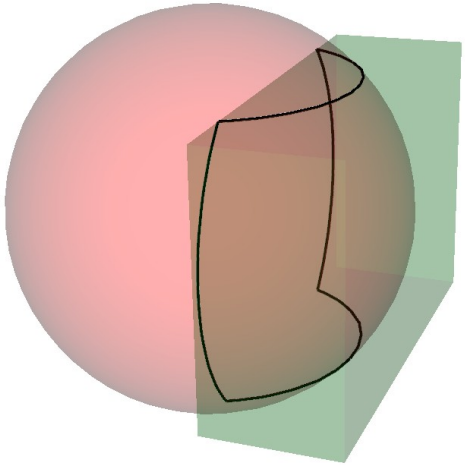
```
1   from vedo import *
2
3   settings.default_backend = "vtk"
4
5   sphere = Sphere().linewidth(1)
6
7   plt = Plotter()
8   plt += sphere
9   plt.show()
10  plt.close()
```

In Jupyter notebooks



Press “h” in rendering window

```
# 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]
```



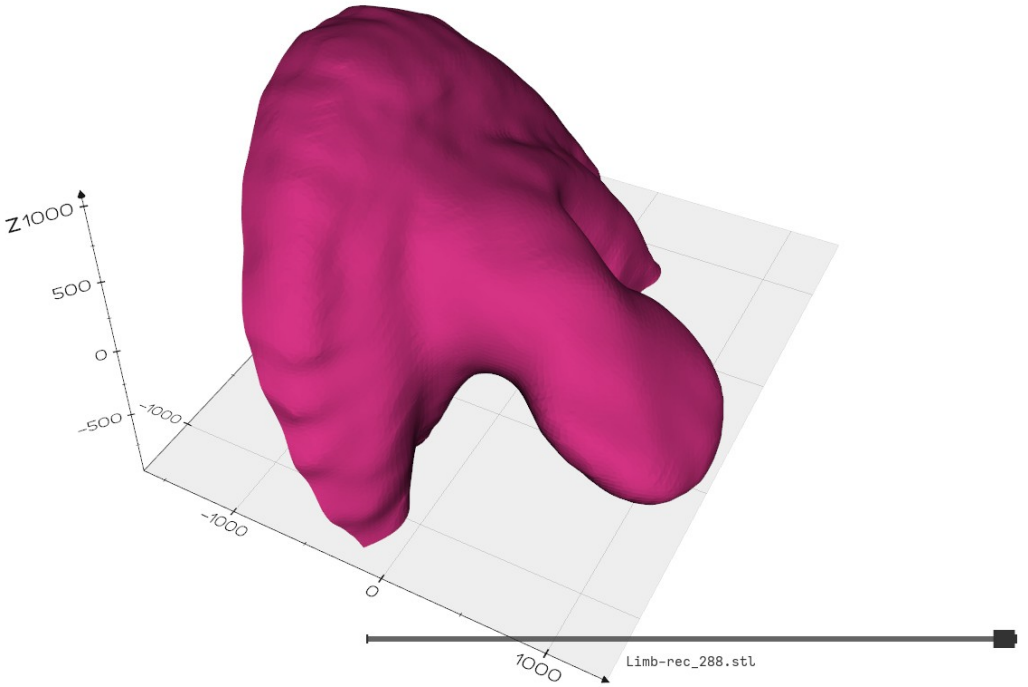
Plotting made simple

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

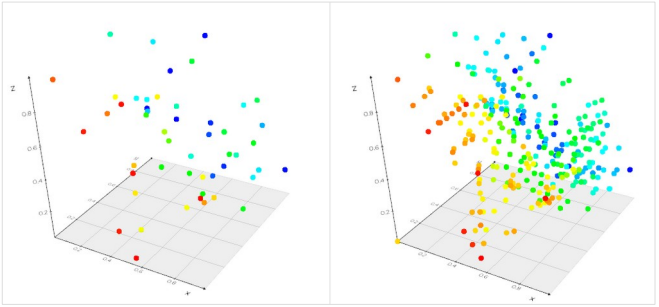
Or in a script:

```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()
```

```
python 01-vis_serie.py
```



Point Clouds and Polygonal Meshes



Create a point cloud and cut it with a plane/mesh

```
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()
```

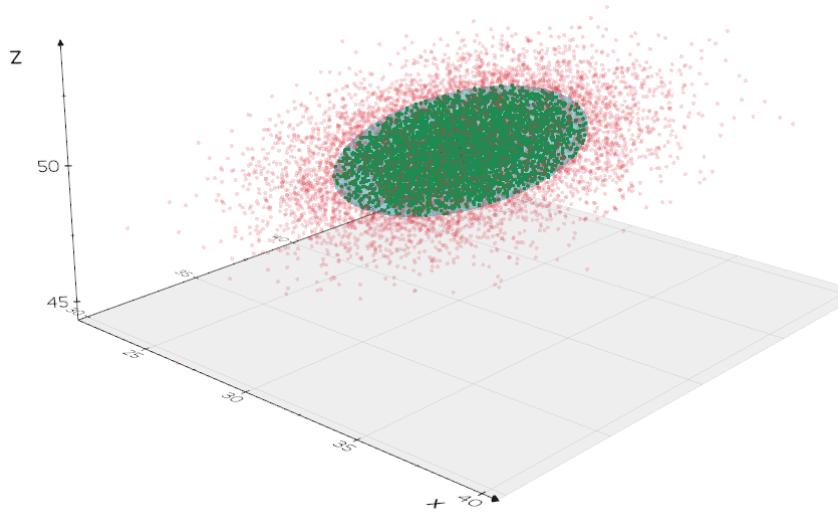
```
pts.cut_with_plane((0.5, 0.5, 0.6))
```

..searching the [API docs]

Fit, count and show a PCA ellipsoid in 3D

```python
from vedo import *

settings.use_depth_peeling = True

pts = Points(np.random.randn(10_000, 3)*[3,2,1])
pts.rotate_z(45).rotate_x(20).shift([30,40,50])

elli = pca_ellipsoid(pts, pvalue=0.50) # 50% of points inside

ids  = elli.inside_points(pts, return_ids=True)
pts.print()  # a new "IsInside" array now exists in pts.pointdata
pin = pts.vertices[ids]
print("inside  points  #", len(pin))

# Create an inverted mask instead of calling inside_points(invert=True)
mask = np.ones(pts.npoints, dtype=bool)
mask[ids] = False
pout = pts.vertices[mask]
print("outside  points #", len(pout))

# Extra info can be retrieved with:
print("axis 1 size:", elli.va)
print("axis 2 size:", elli.vb)
print("axis 3 size:", elli.vc)
print("axis 1 direction:", elli.axis1)
print("axis 2 direction:", elli.axis2)
print("axis 3 direction:", elli.axis3)
print("asphericity:", elli.asphericity(), '+-', elli.asphericity_error())

a1 = Arrow(elli.center, elli.center + elli.axis1)
a2 = Arrow(elli.center, elli.center + elli.axis2)
a3 = Arrow(elli.center, elli.center + elli.axis3)
triad = Assembly(a1, a2, a3) # let's group them

show(
     elli, triad,
     Points(pin).c("green4"),
     Points(pout).c("red5").alpha(0.2),
     __doc__,
     axes=1,
).close()
```

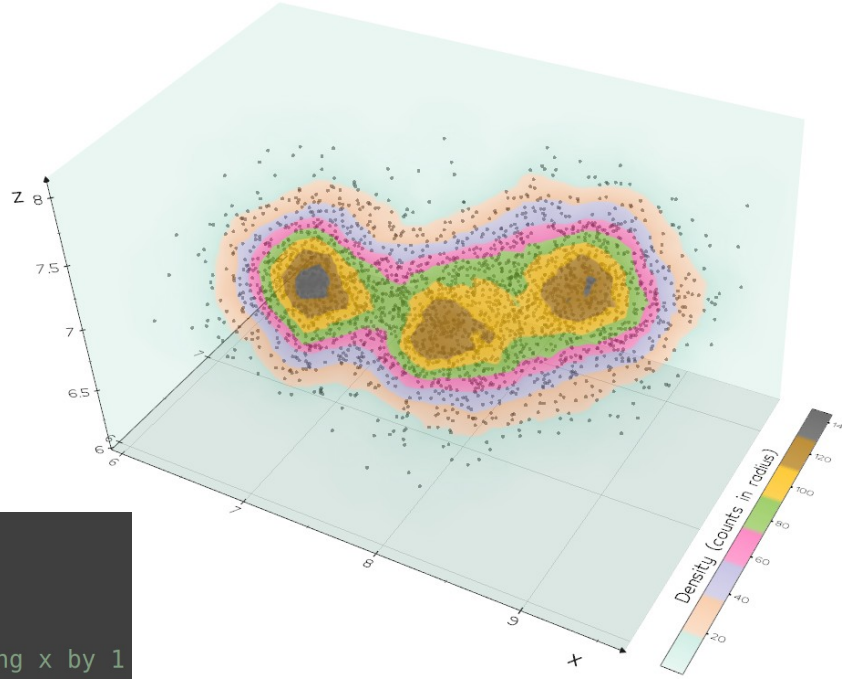
```
> vedo -r pca_ellipsoid
```



Most vedo objects can contain multiple scalar, vectorial and tensorial data associated to vertices and/or cells

Compute point density

..from a point cloud to a TIFF stack



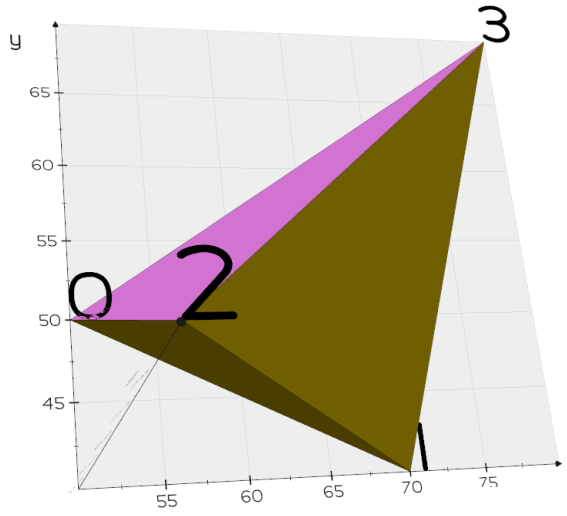

```
> vedo --run plot_density3d
```

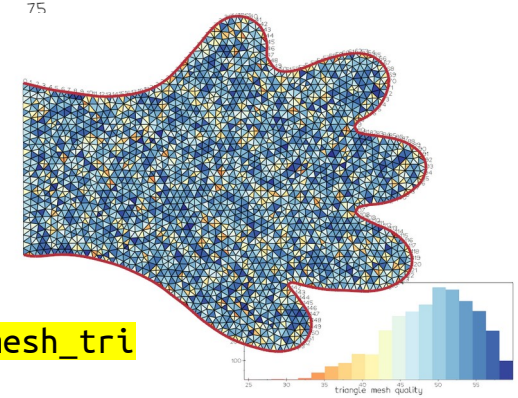
```python
from vedo import *

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()
```

Build a polygonal mesh from vertices and faces

```
> vedo --run buildmesh
```

```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)]
cells = [(0,1,2), (2,1,3), (1,0,3)] # cells same as faces

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

# 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.labels2d('pointid')

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

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





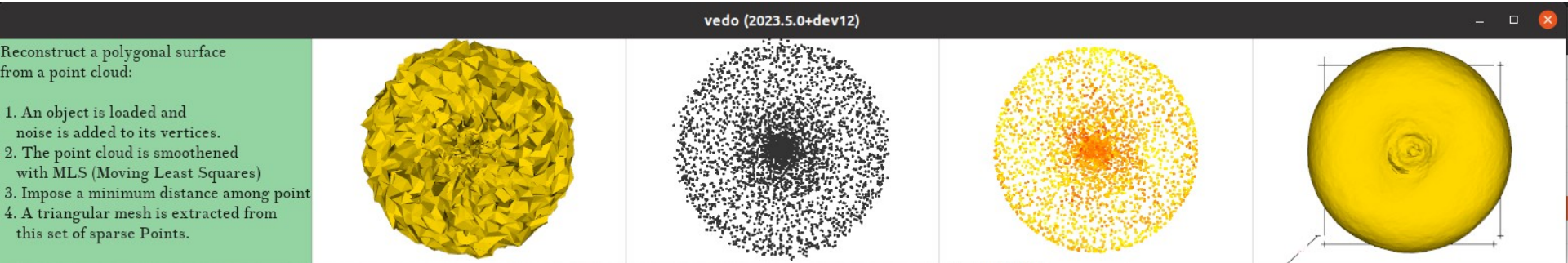
..or from an outline: `vedo --run line2mesh_tri`

Reconstruct a polygonal mesh from a point cloud

```
> vedo --run recosurface
```

...including smoothing of the point cloud
with MLS (moving least squares)
...visualize objects in sync'ed renderers

```python
from vedo import dataurl, printc, Plotter, Points, Mesh, Text2D


plt = Plotter(shape=(1,5))
plt.at(0).show(Text2D(__doc__, s=0.75, font='Theemim', bg='green5'))

# 1. load a mesh
mesh = Mesh(dataurl+"apple.ply").subdivide()
plt.at(1).show(mesh)

# Add noise
pts0 = Points(mesh, r=3).add_gaussian_noise(1)
plt.at(2).show(pts0)

# 2. Smooth the point cloud with MLS
pts1 = pts0.clone().smooth_mls_2d(f=0.8)
printc("Nr of points before cleaning nr. points:", pts1.npoints)

# 3. Impose a min distance among mesh points
pts1.subsample(0.005)
printc("            after  cleaning nr. points:", pts1.npoints)
plt.at(3).show(pts1)

# 4. Reconstruct a polygonal surface from the point cloud
reco = pts1.reconstruct_surface(dims=100, radius=0.2).c("gold")
plt.at(4).show(reco, axes=7, zoom=1.2)

plt.interactive().close()
```



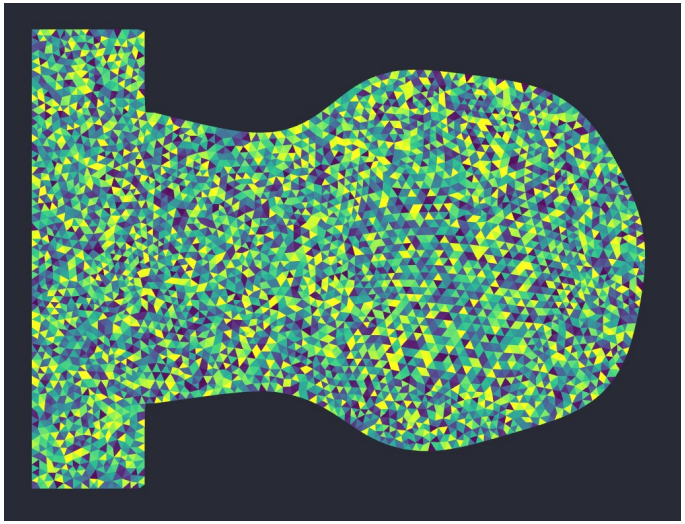
Add gene data associated to mesh cells (faces)

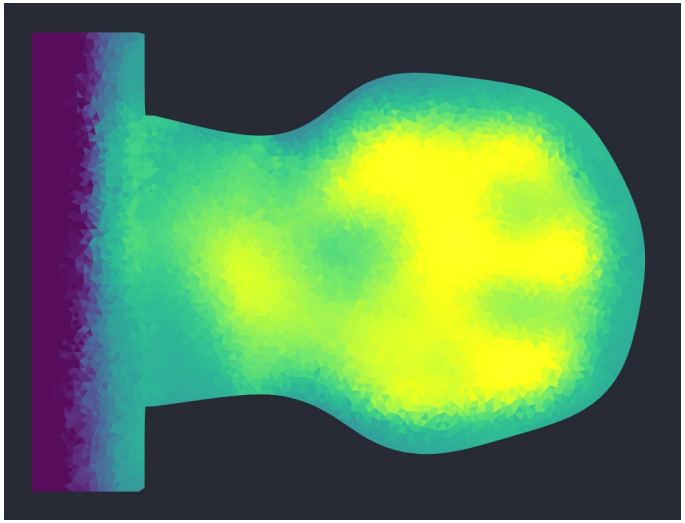
```
msh = Mesh([verts, faces]).linewidth(1)
```

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

Using file `../data/sox9_data/gene_data.npy`

```
> python 02-gene_mesh.py
```





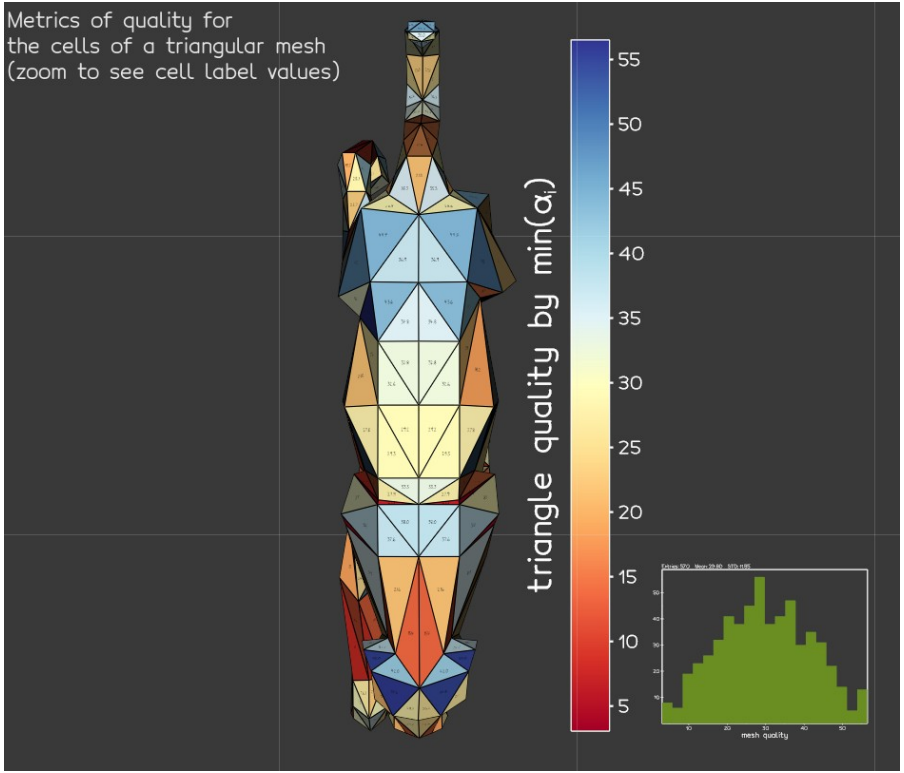
Compute various properties of the mesh

E.g.: face “quality”, scalar gradients, curvatures…

Show labels for the associated data

```
> vedo --run meshquality
```

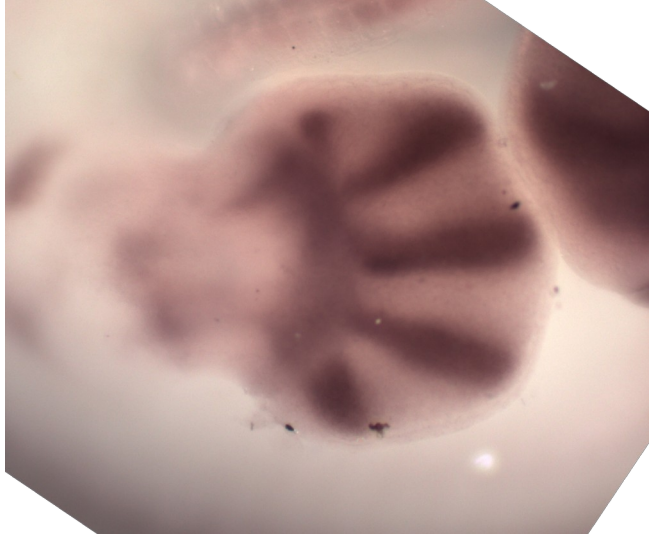
```
from vedo import dataurl, Mesh, show
from vedo.pyplot import histogram

mesh = Mesh(dataurl + "panther.stl").compute_normals().linewidth(0.1).flat()

# generate a numpy array for mesh quality
mesh.compute_quality(metric=6)
mesh.cmap("RdYlBu")

hist = histogram(mesh.celldata["Quality"], xtitle="mesh quality", ac="w")
# make it smaller and position it, use_bounds makes the cam
# ignore the object when resetting the 3d qscene
hist.scale(0.6).pos(40, -53, 0).use_bounds(False)

# add a scalar bar for the active scalars
mesh.add_scalarbar3d(c="w", title="triangle quality by min(:alpha_i )")

# create numeric labels of active scalar on top of cells
labs = mesh.labels(on="cells", precision=3, scale=0.4, font="Quikhand", c="black")

cam = dict(
    pos=(59.8, -191, 78.9),
    focal_point=(27.9, -2.94, 3.33),
    viewup=(-0.0170, 0.370, 0.929),
    distance=205,
    clipping_range=(87.8, 355),
)

show(mesh, labs, hist, __doc__, bg="bb", camera=cam, axes=11).close()
```



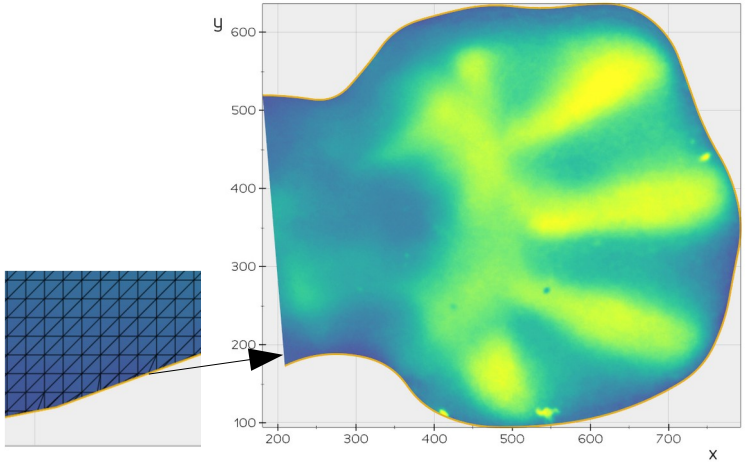
Make a Mesh from a JPG image

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

```python
from vedo import Image, settings, show
from vedo.applications import SplinePlotter

settings.default_backend = "vtk" # for jupyter notebooks

pic = Image("../data/sox9_data/sox9_exp.jpg").bw()

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

# create a mesh from the image
msh = pic.tomesh()

print("Cutting using outline... (please wait)")
cut_msh = msh.clone().cut_with_point_loop(outline)
print(cut_msh)

cut_msh.pointdata.select("RGBA")
cut_msh.cmap("viridis_r").add_scalarbar3d("Sox9 Level")

show(cut_msh, outline, axes=1, zoom='tight').close()
```



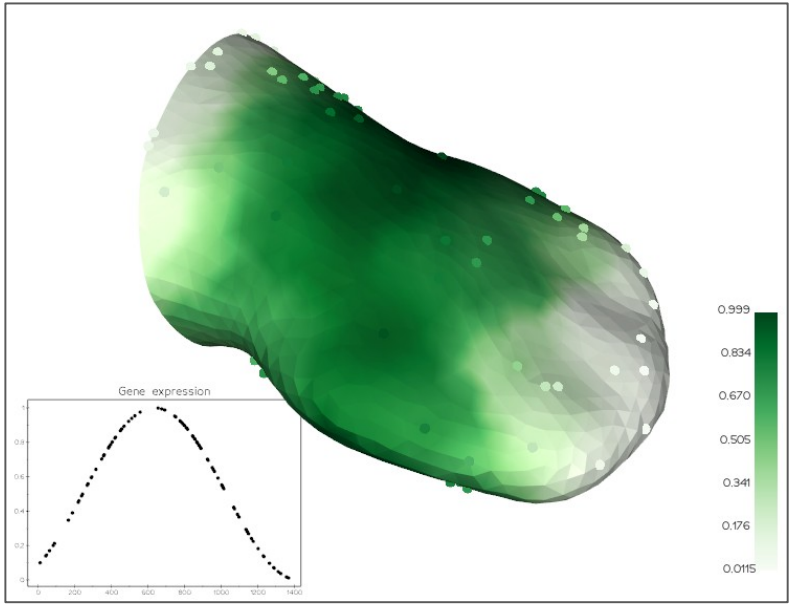
```
> python 03-interpolate_scalars2d.py
```



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)
```



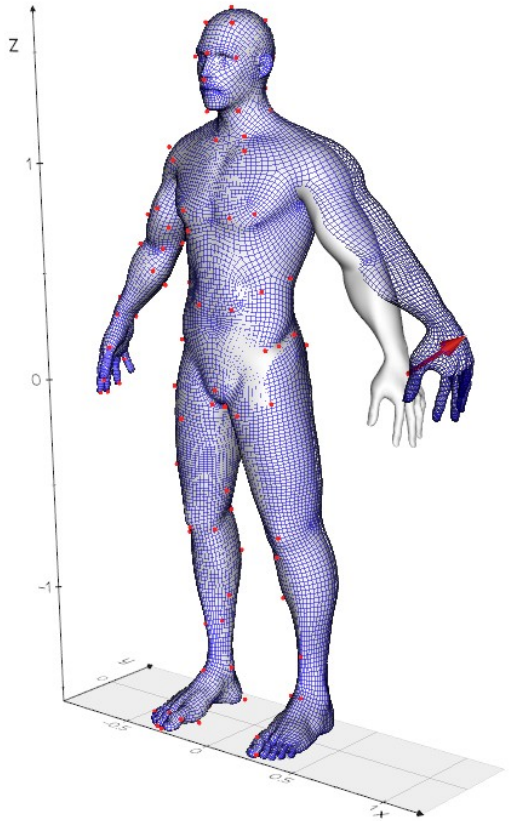

```
> python 04-interpolate_scalars3d.py
```

Warp a Mesh (non-linear registration)

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

```
> python 05-warp_mesh.py
```

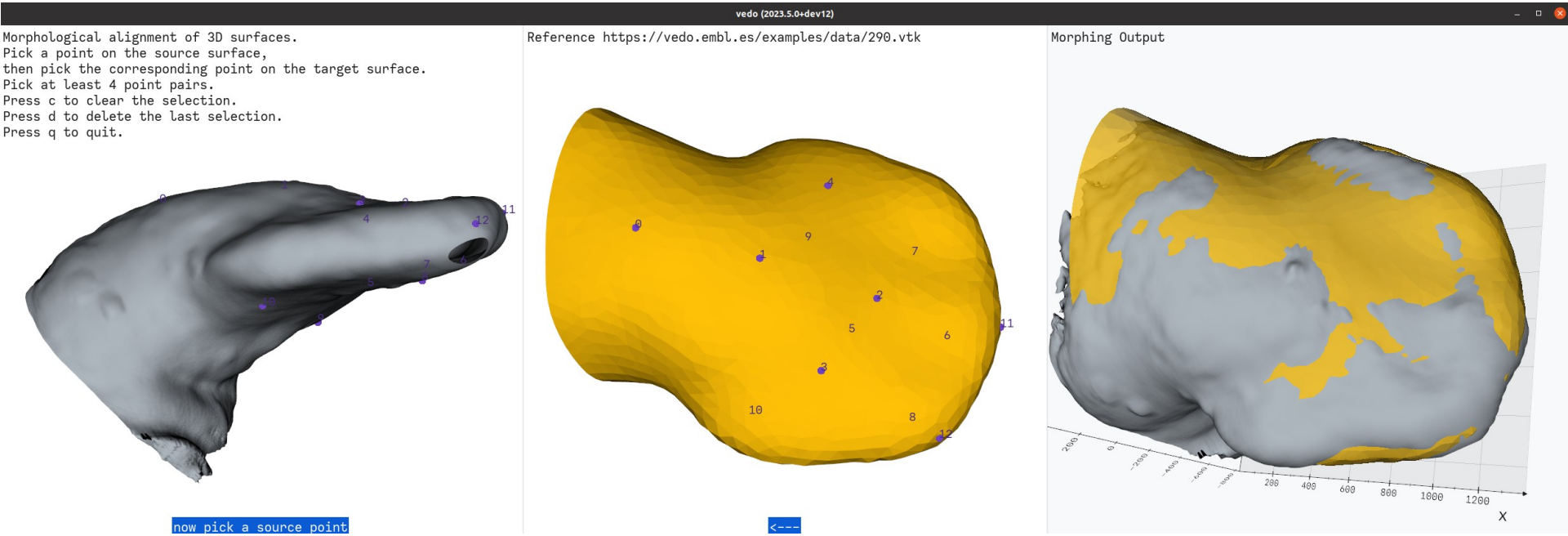
```python
from vedo import dataurl, Mesh, Arrows, show

# 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)
```



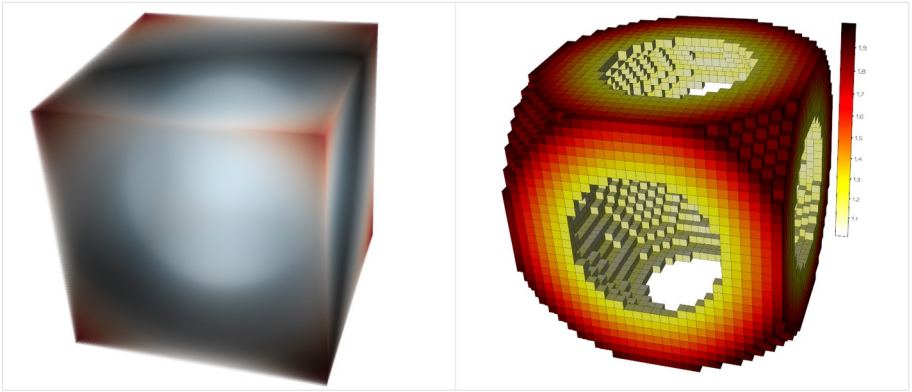
Warp a Mesh interactively

Create a small custom application (<100 lines of code)

```
> vedo --run warp
```



Volumes
(eg TIFF stacks)

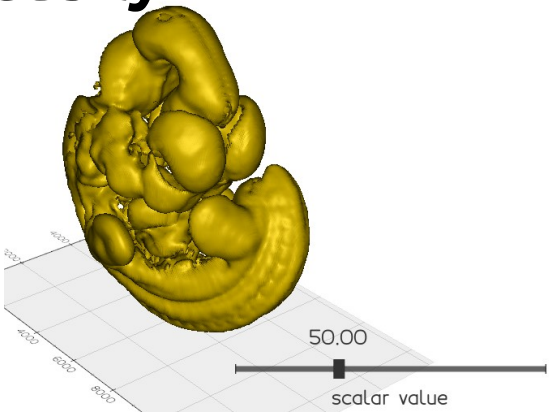


```
> vedo --run numpy2volume1
```

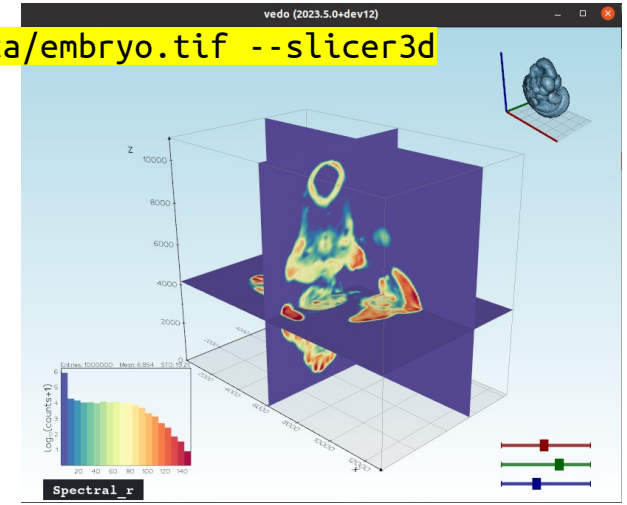
Visualize your data easily

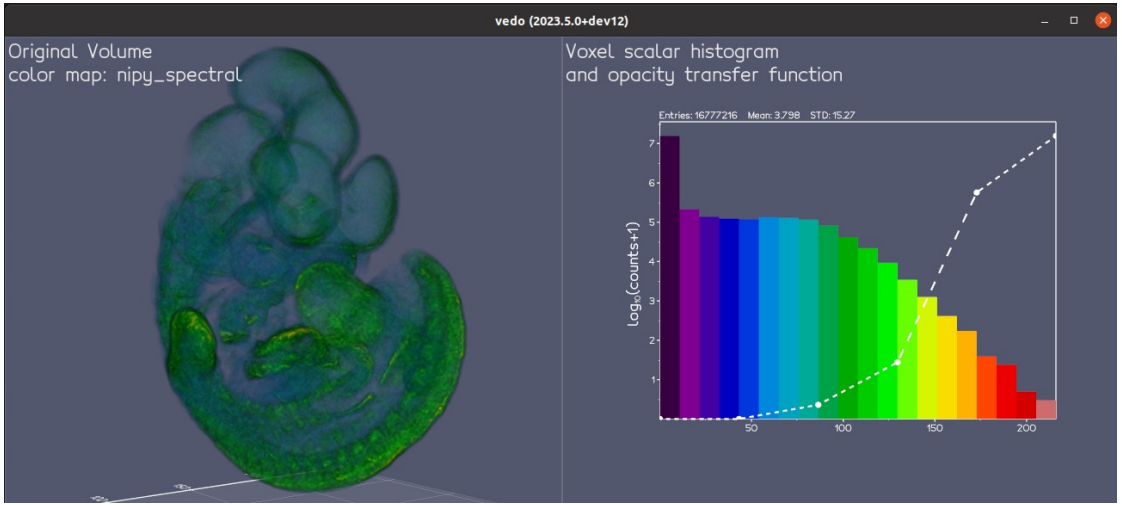
```
> vedo ../data/embryo.tif
```

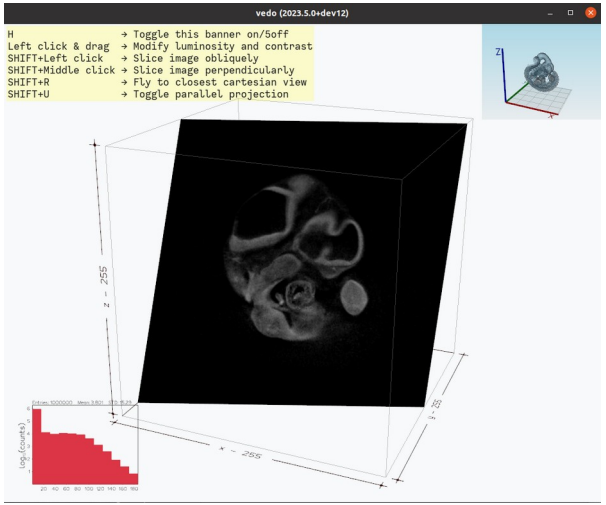
```
> vedo --run read_volume1
> vedo --run read_volume2
> vedo --run read_volume3
```




```
> vedo ../data/embryo.tif --slicer3d
```



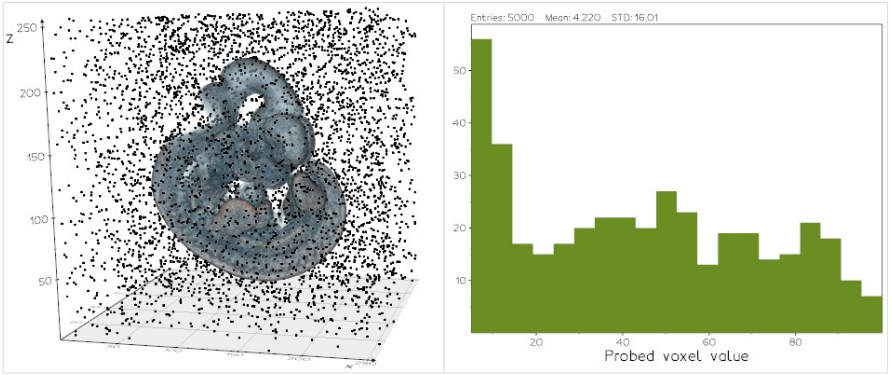


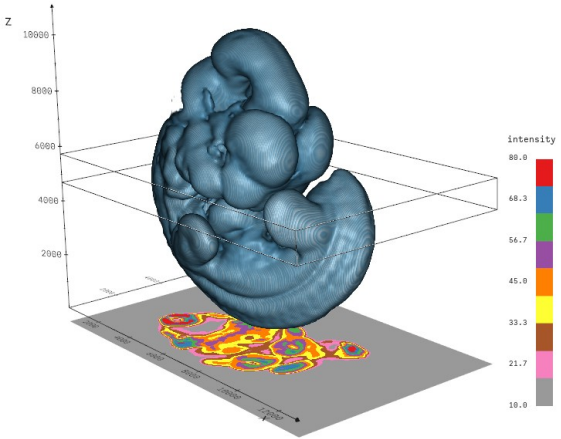


Analyze your data easily

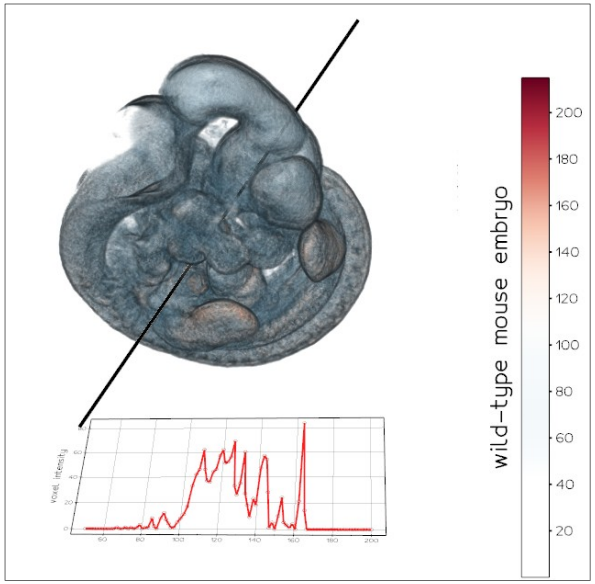
Many different operations are possible!
(eg. FFT, gradients, smoothing...)

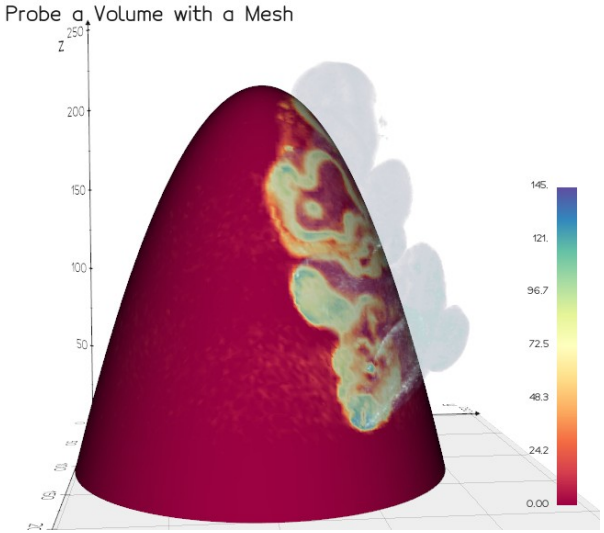
```
> vedo --run probe
```



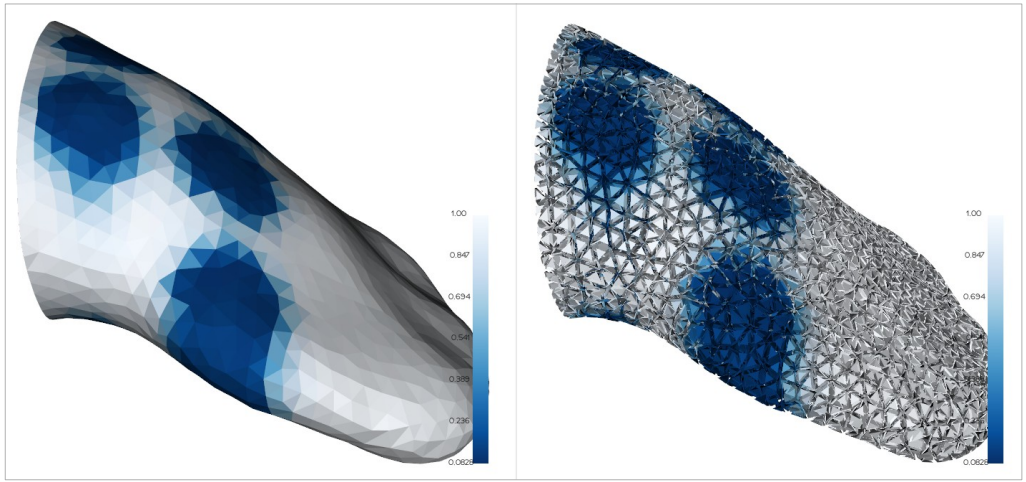



```
> vedo --run slab_vol
```

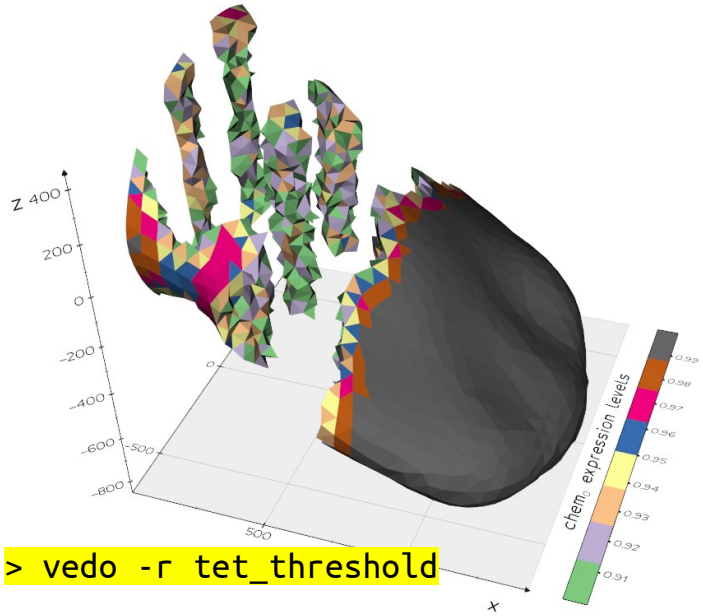




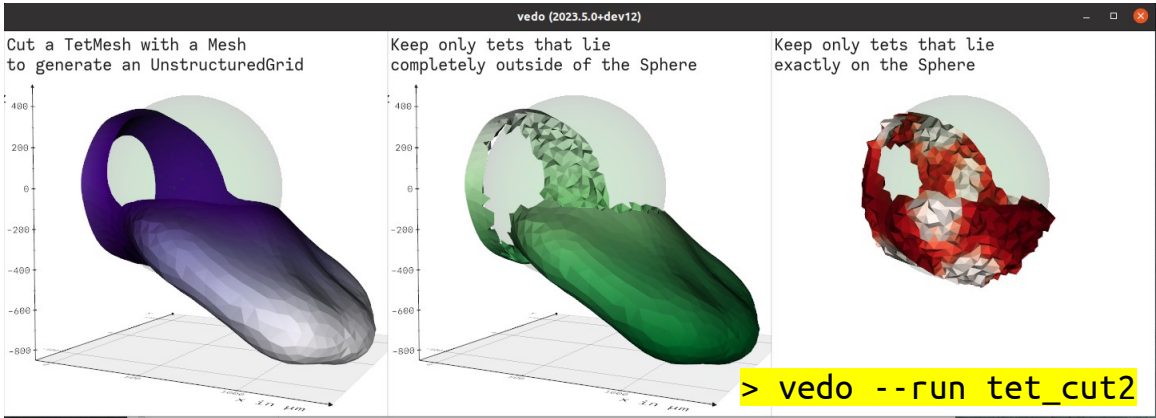
Generate and work with tetrahedral meshes



```
> vedo --run tet_astyle
```



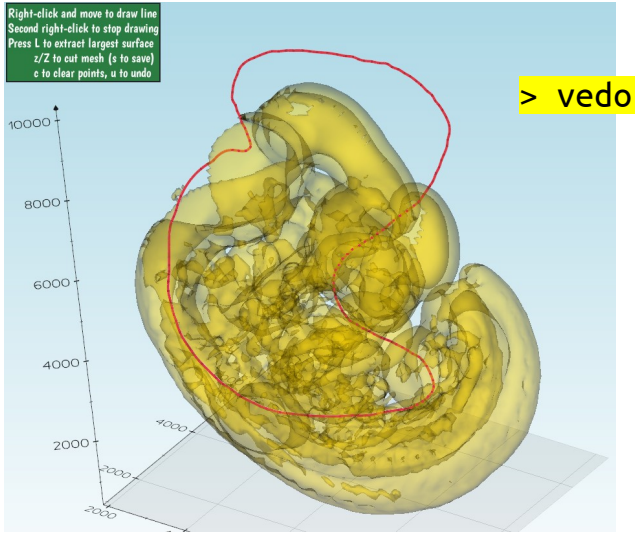

```
> vedo -r tet_threshold
```




```
> vedo --run tet_cut2
```

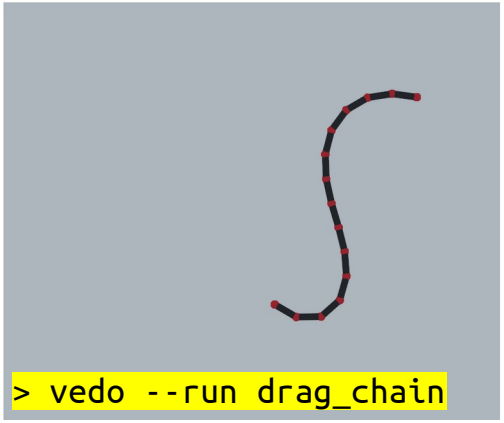
Animation and interaction

Can create different types of interaction “events”
with mouse or keyboard in callbacks
to generate window interactivity




```
> vedo --run cut_freehand
```

Forward kinematics: hover the mouse to drag the chain

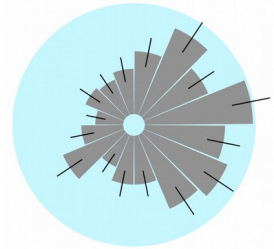


```
> vedo --run drag_chain
```

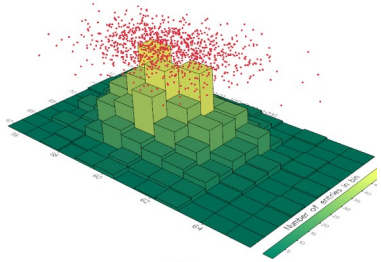
```python
from vedo import Plotter, versor, Plane, Line

n = 15  # number of points
l =  3  # length of one segment

def func(evt):
    if not evt.object:
        return
    coords = line.vertices
    coords[0] = evt.picked3d
    for i in range(1, n):
        v = versor(coords[i] - coords[i-1])
        coords[i] = coords[i-1] + v * l
    line.vertices = coords  # update positions
    nodes.vertices = coords
    plt.render()

surf = Plane(s=[60, 60])
line = Line([l*n/2, 0], [-l*n/2, 0], res=n, lw=12)
line.render_lines_as_tubes()
nodes= line.clone().c('red3').point_size(15)

plt = Plotter()
plt.add_callback("on mouse move please call", func)
plt.show(surf, line, nodes, __doc__, zoom=1.3)
plt.close()
```

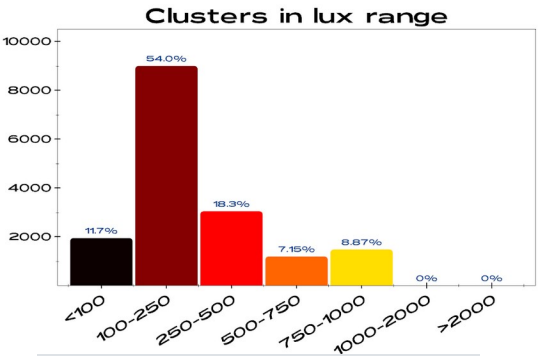
Save and read back full 3D scene snapshots:

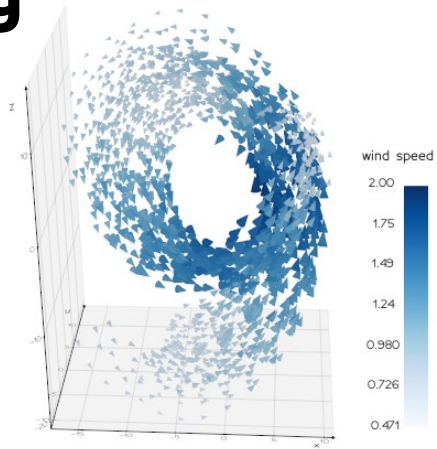
```
> vedo https://vedo.embl.es/examples/geo_scene.npz
```

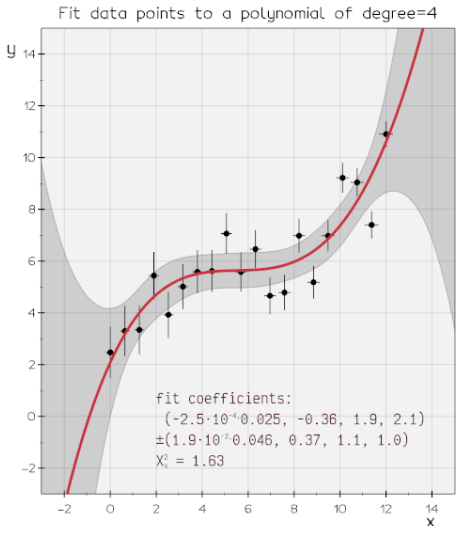


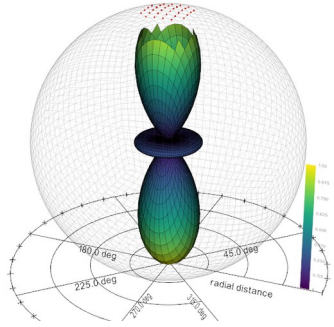
2D/3D Plotting

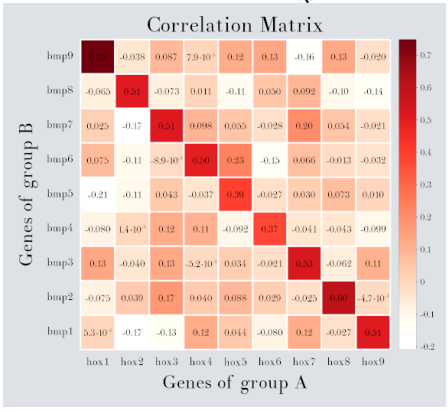


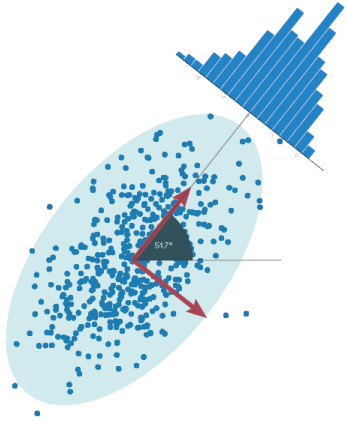


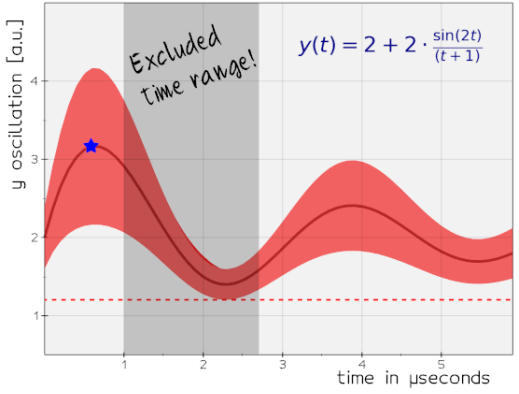


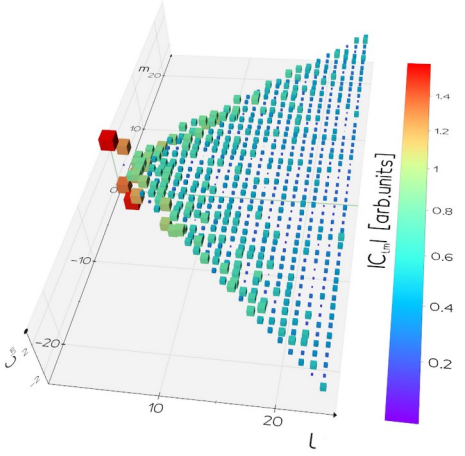












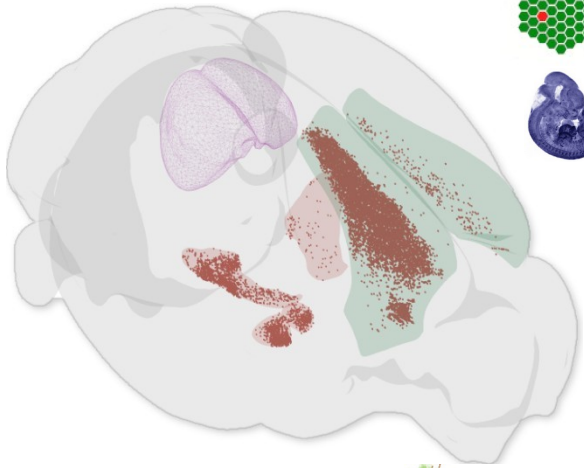
Conclusion

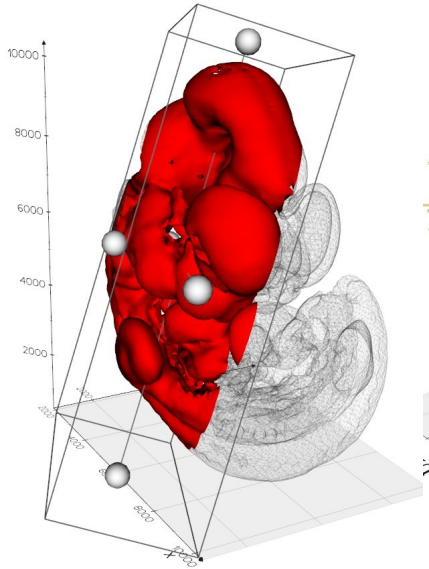
- Proved very useful in diverse applications
- Documented API with many examples
- **Happy to offer support!**

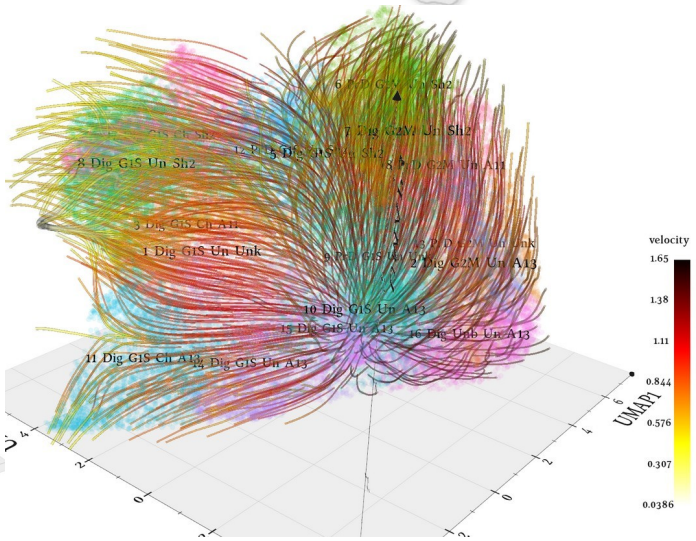
marco.musy@embl.es

https://vedo.embl.es/

"I rarely found a better documented and more accessible package than vedo"
F. Claudi, Sainsbury Wellcome Centre







EMBL, EPUG Dec. 2023