PoL
Physics of Life
TU Dresden

CENTER FOR
SYSTEMS BIOLOGY
DRESDEN

Basic instance segmentation

Johannes Soltwedel & Marco Musy

With materials from
Robert Haase
Till Korten

@jm_mightypirate

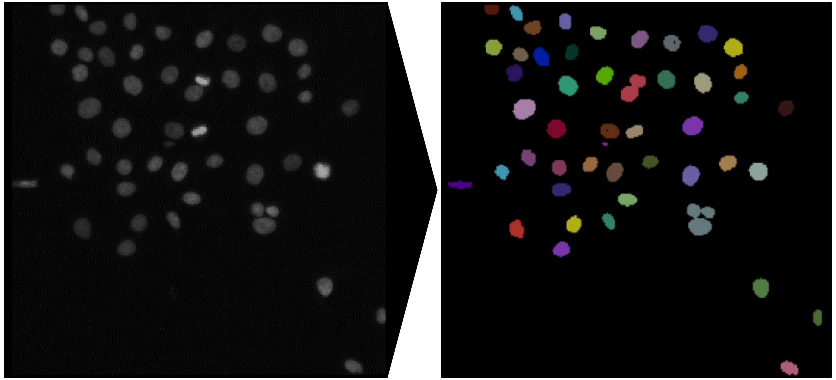
Segmentation

Aim:

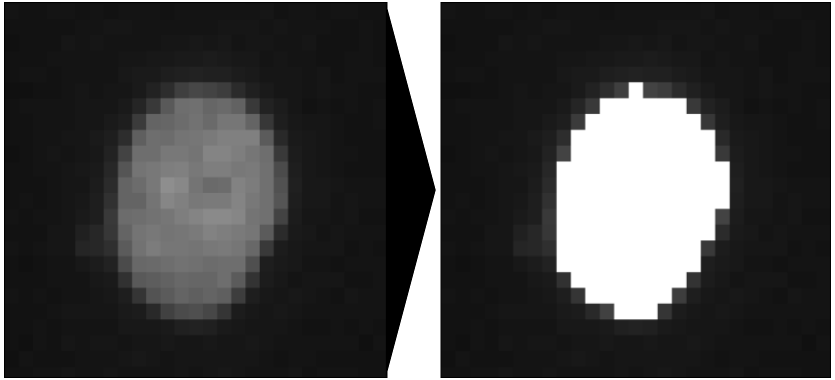
Separate background from foreground

Vocabulary:

- **Segmentation:**
  → Assigning a meaningful *label* to each pixel
  → Segmentation is a *classification* problem
- **Semantic segmentation:**
  Differentiate pixels into multiple *classes* (e.g., membrane, nucleus, cytosol, etc.)
- **Instance segmentation:**
  Differentiate multiple occurrences of the same class into separate instances of this class (e.g., separate *label* for each cell in image)



Instance segmentation

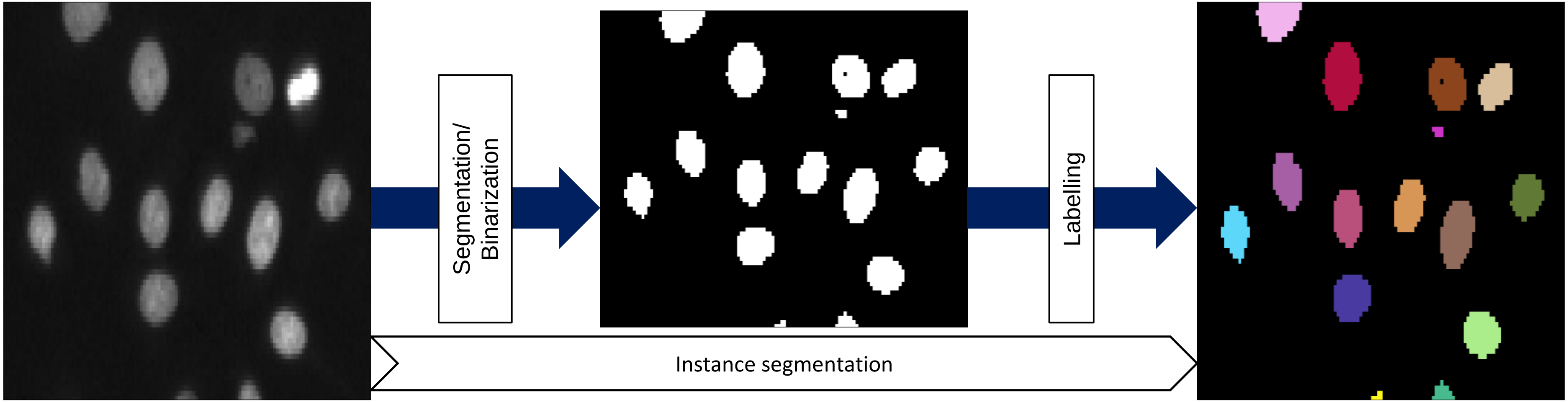


Semantic segmentation

Segmentation and labelling

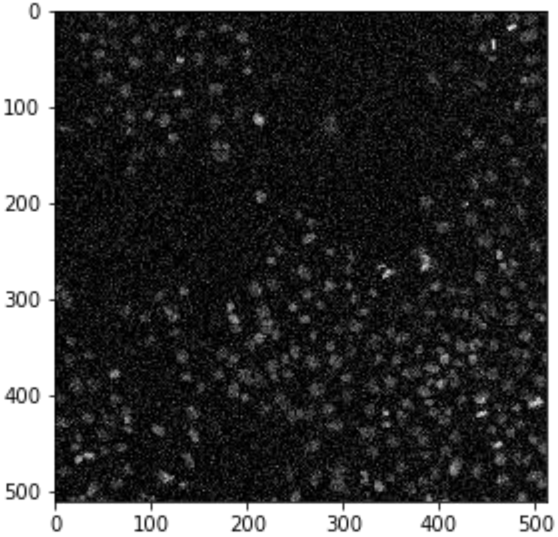
Analyzing properties (*features*) of individual objects in images requires instance segmentation

- Methods
  - Thresholding + connected components labeling
  - Spot detection + seeded watershed
  - Edge detection based
  - Machine learning

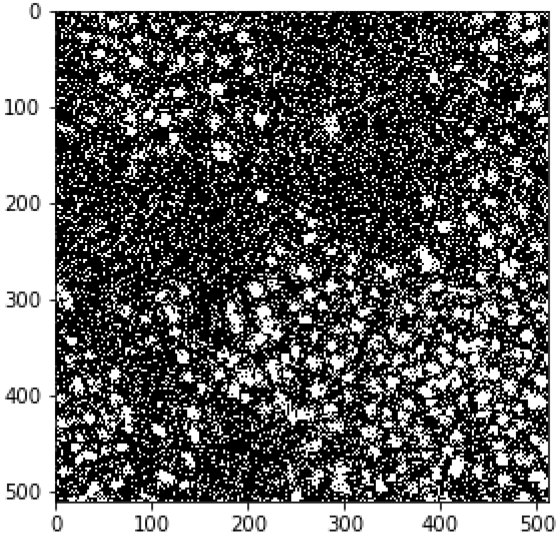


Reminder: pre-processing!

- Before we can create masks, we need to pre-process images:
  - Noise removal
  - Background subtraction



Noisy image



Thresholded image

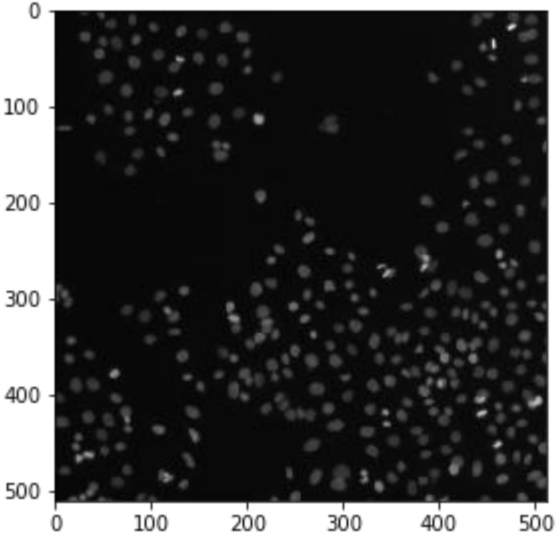
```
filtered = filters.median(image)
```

Image filtering *filters* relevant information for subsequent operations from the image!

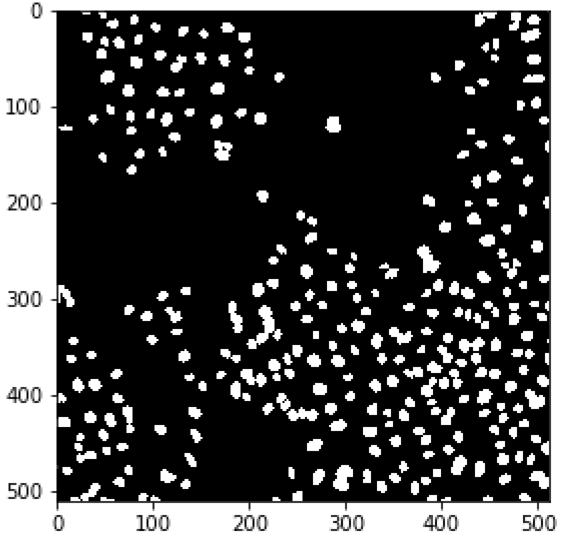
https://scikit-image.org/docs/stable/api/skimage.data.htm

Reminder: pre-processing!

- Before we can create masks, we need to pre-process images.
  - Noise removal
  - Background subtraction



Filtered image



Thresholded image

```
filtered = filters.median(image)
```

Image filtering *filters* relevant information for subsequent operations from the image!

@jm_mightypirate

https://scikit-image.org/docs/stable/api/skimage.data.htm

Thresholding in Python

- Applying a threshold to an image requires to compare every pixel to the threshold value
- We can compare values in Python with:

```
a = 5
b = 6
print(a > b)
print(a < b)
print(a==b)
```

```
False
True
False
```

```
image > threshold
```

```
array([[False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]])
```

In this case, "image" is a *numpy array* → some operations are automatically applied to every pixel!

- We can then simply store the output of this element-wise comparison in a new variable:

```
binary = image > threshold
```

Thresholding with scikit-image

```python
threshold = filters.threshold_otsu(image)
```

- **Otsu-thresholding** (Otsu et Al. 1979): Find threshold so that the summed, weighted variance $Var_{w,sum}$ becomes minimal.

```python
threshold = filters.threshold_mean(image)
```

- **Statistical thresholding**: Pixels above statistical

```python
threshold = filters.threshold_triangle(image)
```

- **Triangle thresholding**: Draw a line between histogram point with max. counts and max. intensity and find point in histogram with maximal distance to this line.

Explore more threshold options in scikit-image with:

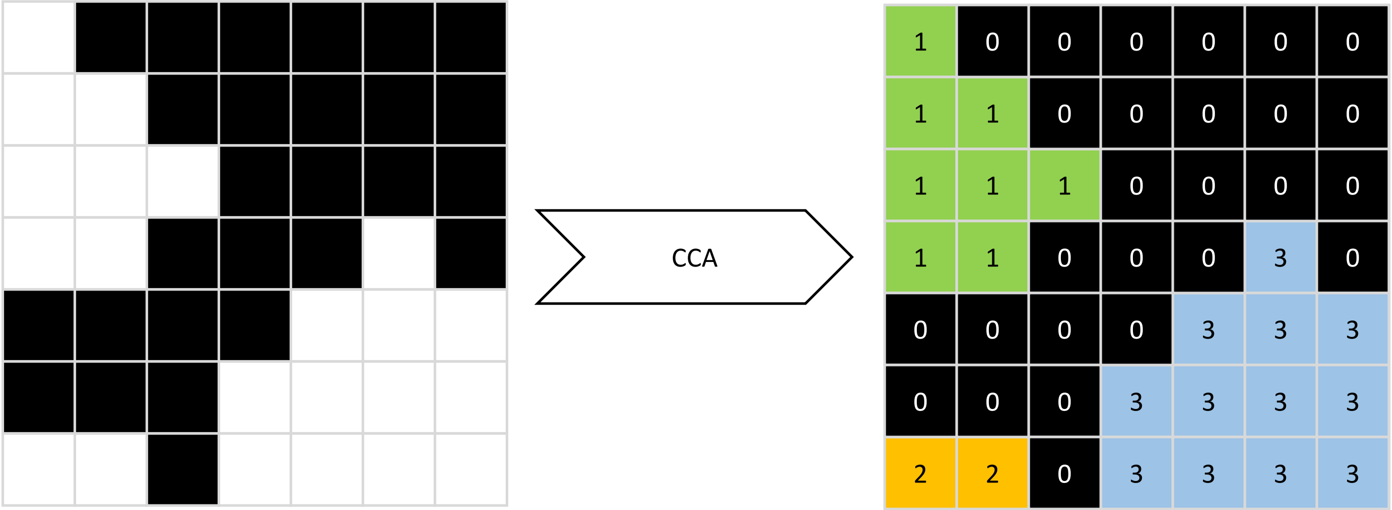
```python
from skimage import filters
```

```python
threshold = filters.threshold_
```

```
f threshold_isodata     function
f threshold_li          function
f threshold_local       function
f threshold_mean        function
f threshold_minimum     function
f threshold_multiotsu   function
f threshold_niblack     function
f threshold_otsu        function
f threshold_sauvola     function
f threshold_triangle    function
```

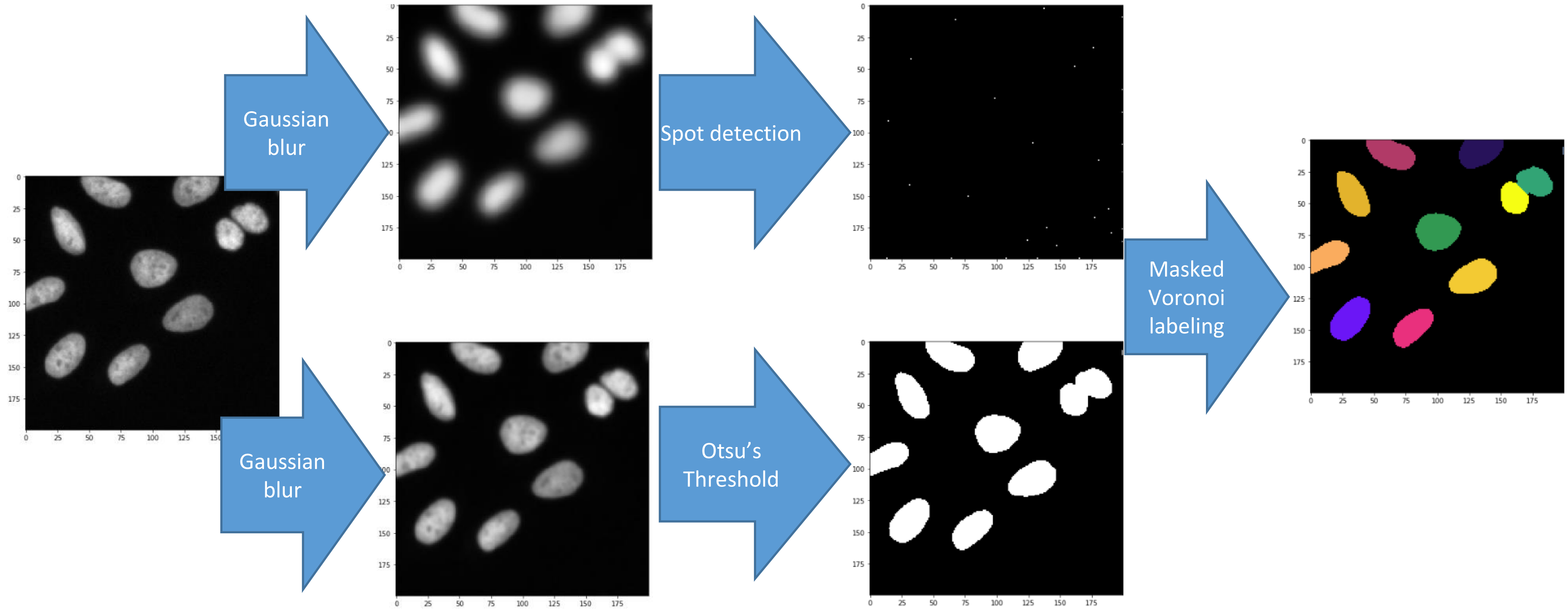
Connected component labelling

- In order to allow the computer differentiating objects, connected component analysis (CCA) is used to mark pixels belonging to different objects with different numbers
- Background pixels are marked with 0.
- The maximum intensity of a labelled map corresponds to the number of objects.



Voronoi-Otsu-Labeling

- Combination of Gaussian blur, Otsu's Threshold and Voronoi-labeling



@jm_mightypirate

https://haesleinhuepf.github.io/BioImageAnalysisNotebooks/20_image_segmentation/11_voronoi_otsu_labeling.html

Mesh processing

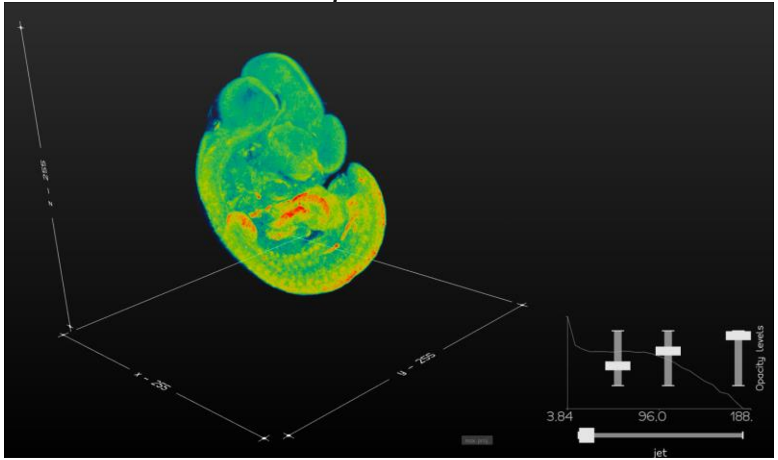
Johannes Soltwedel & Marco Musy

With materials from
Robert Haase
Till Korten

@jm_mightypirate

vedo

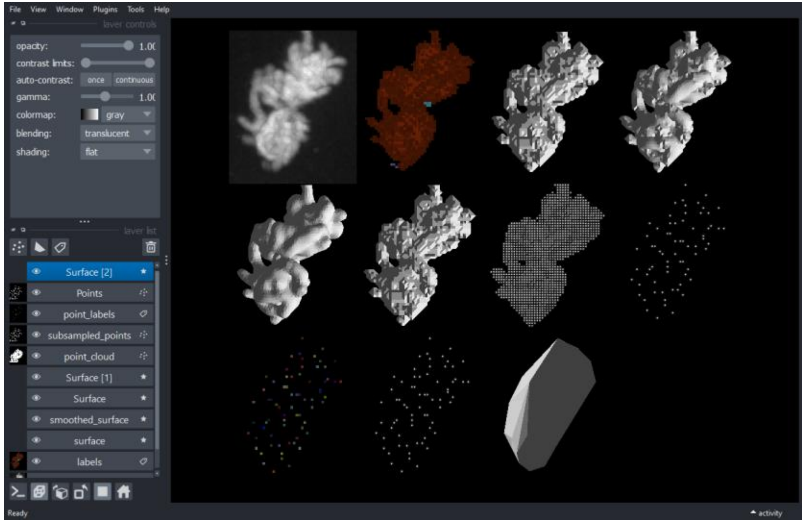
A python module for scientific analysis and visualization of эd objects



https://vedo.embl.es

Napari-process-points-and-surfaces

(Napari wrapper for vedo)



https://napari-hub.org/napari-process-points-and-surfaces

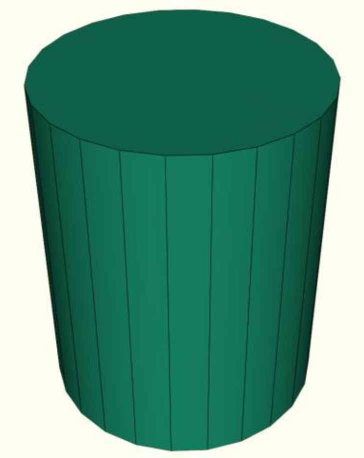
Why?

`matplotlib` is not very useful in 3D

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

```python
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()
```

```python
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!

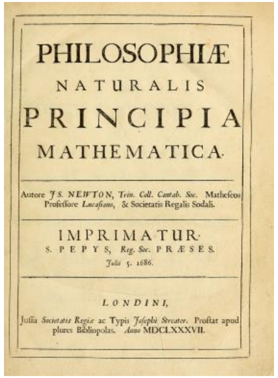
R. de Bruin, Delft Univ. of Tech

Where?

vedo sits somewhere in here

Preprocessing
Raw Data

Postprocessing
Data analysis



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.

TB GB MB

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

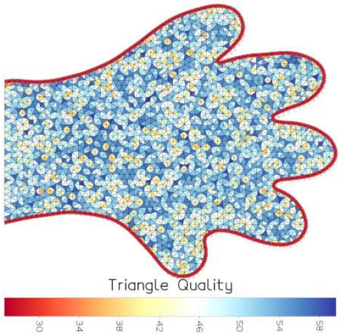
"A handy day-to-day tool for the researcher"

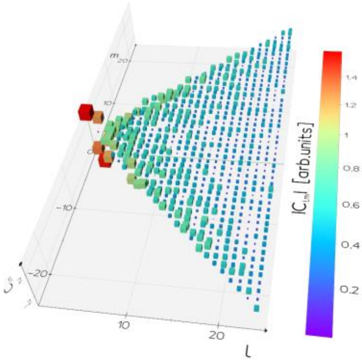
It can prove useful with any type of data having a spatio-temporal structure.

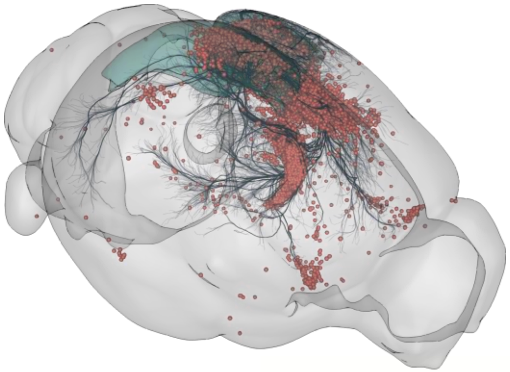
Icons by Iori Johnstor and ProSymbols from Noun Project

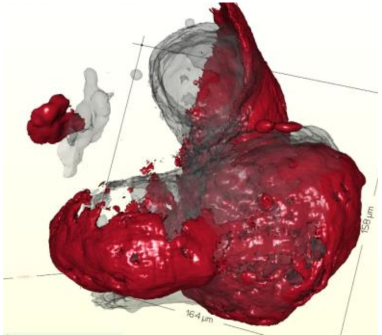
What can *you* do with it?

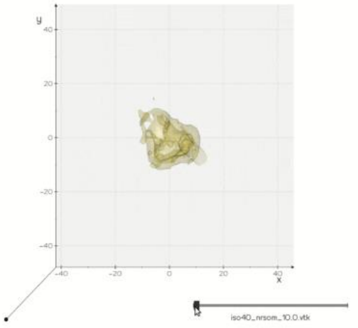
- Work with polygonal meshes and point clouds
- Morphometrics (mesh warp, cut, connect, ...)
- Analysis of 3D images and tetrahedral meshes
- 2D/3D plotting and histogramming.
- Integration with other external libraries (`Qt`, `napari`, `trimesh`, `pymeshlab`, `SHTools` ...)
- Jupyter and Colab environments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



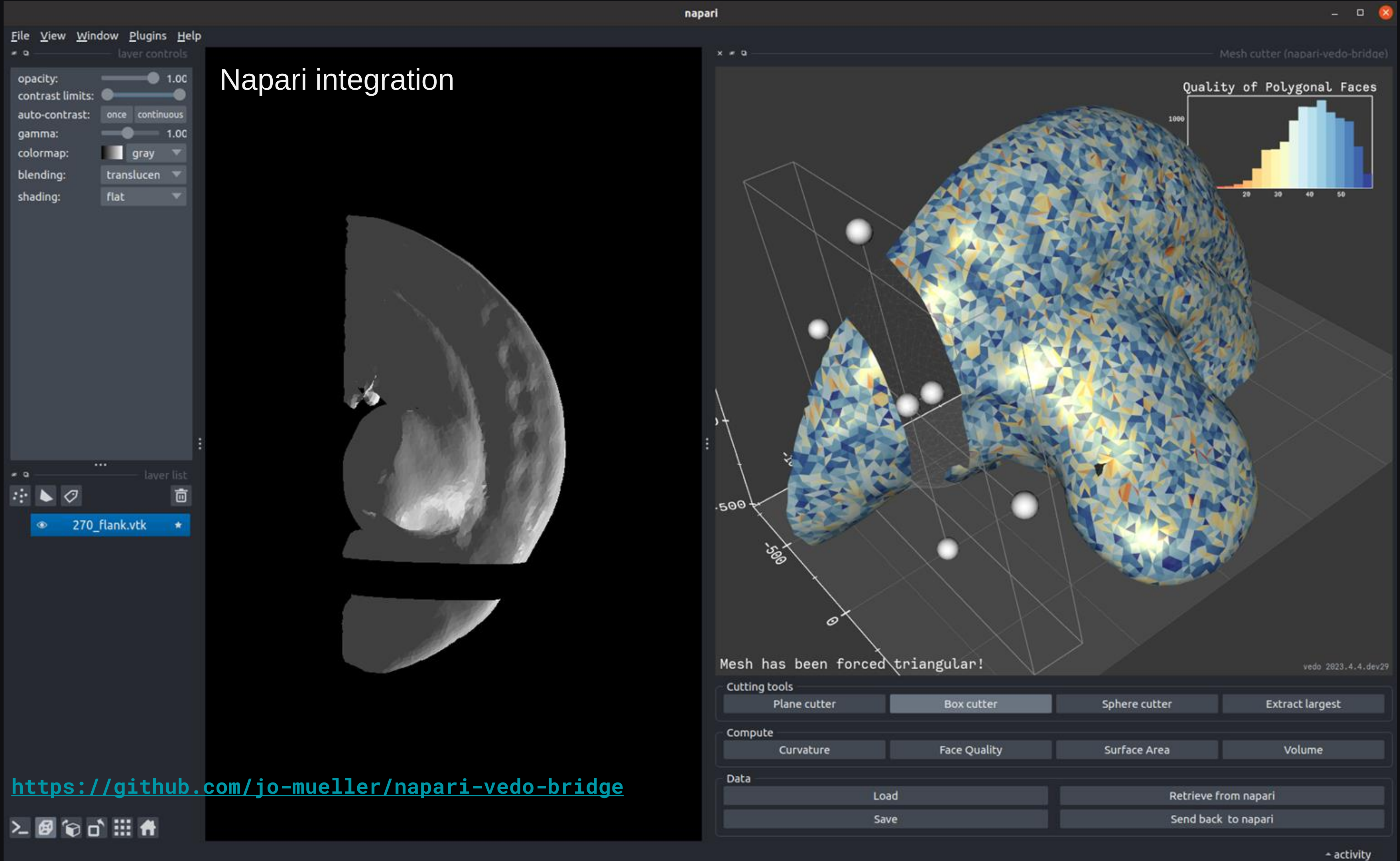








Napari integration



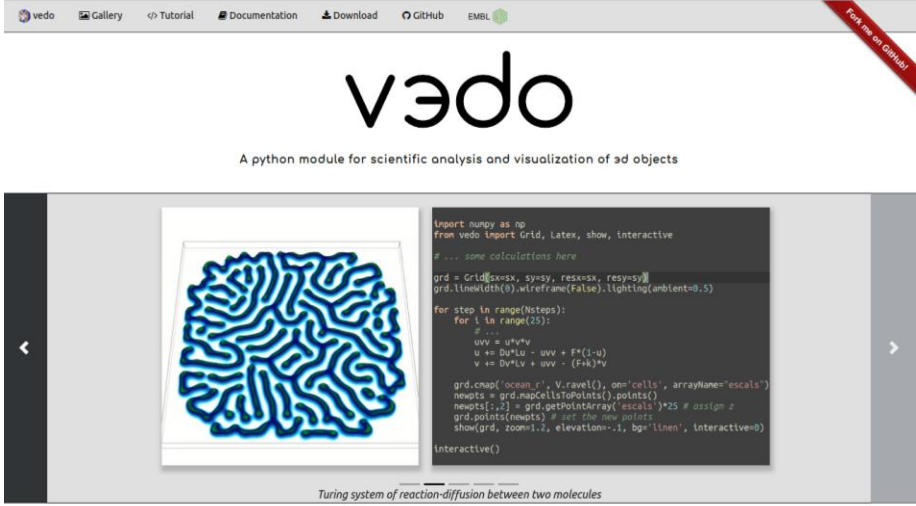
https://github.com/jo-mueller/napari-vedo-bridge

How?

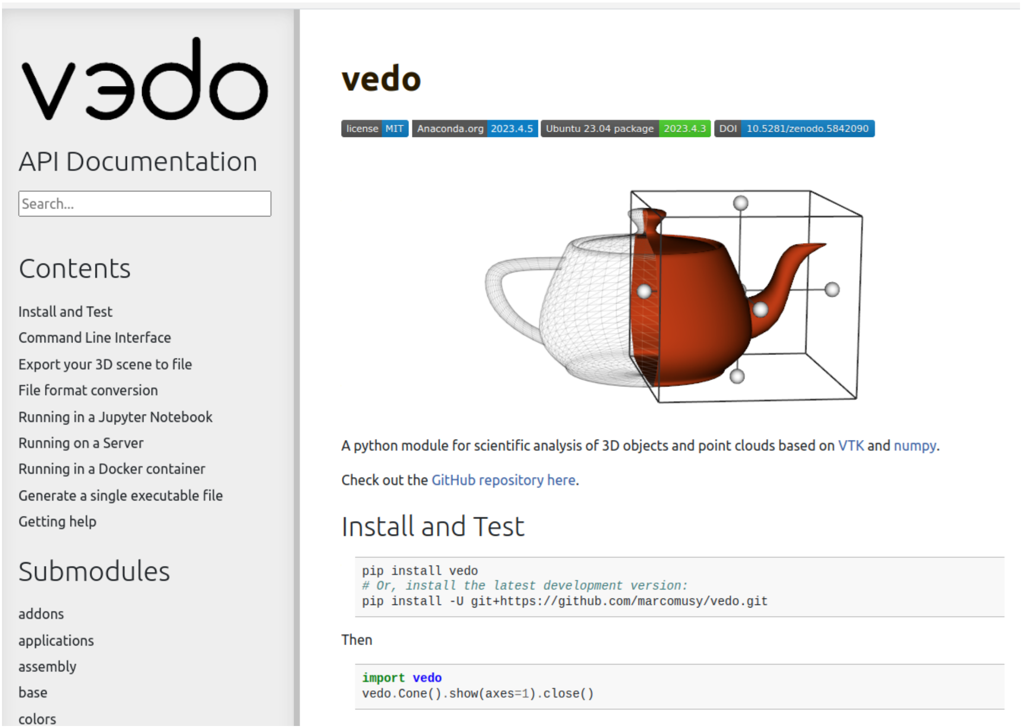
- Install:

  `mamba 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`



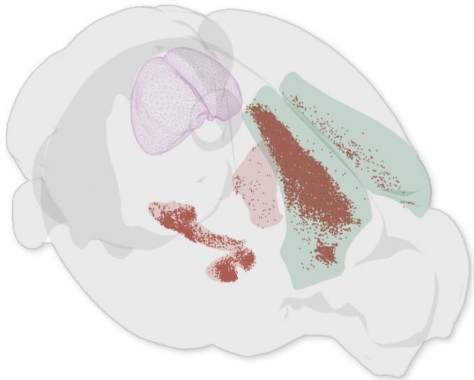
API documentation is found at vedo.embl.es/docs

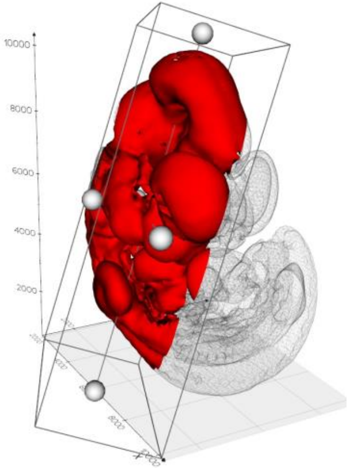


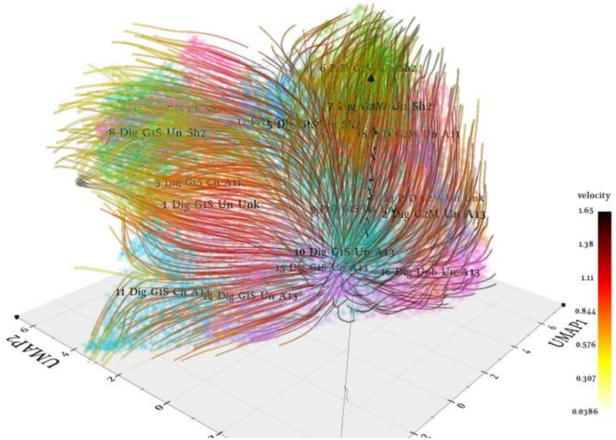
Conclusion

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

  marco.musy@embl.es
  laura.avino@embl.es



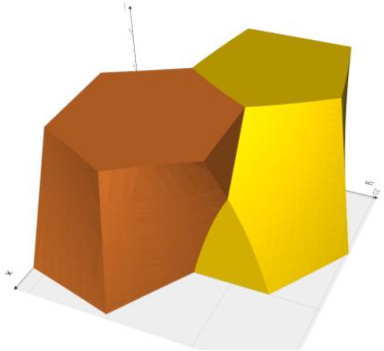




[**https://vedo.embl.es/**](https://vedo.embl.es/)

SHARPE LAB

EMBL
European Molecular Biology Laboratory



vedo practicals

Installing steps

Install dependencies:

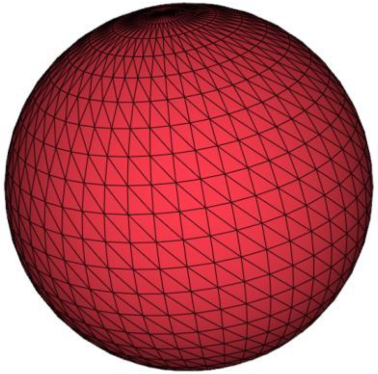
```
> mamba install vedo
> pip install napari-process-points-and-surfaces
```

01-basics.ipynb

Basic Geometric Objects

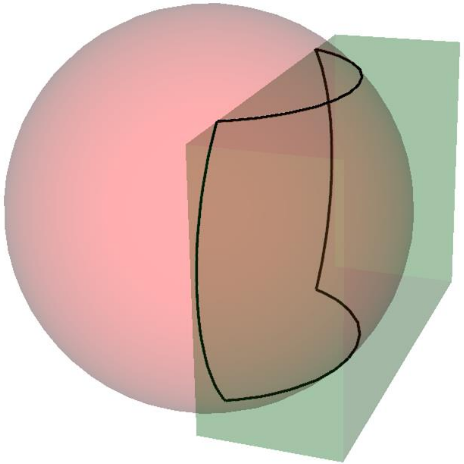
```python
from vedo import *

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]
```



02-chaste_vis.ipynb

Plotting made simple

Example from Tuesday Chaste tutorial

```
> vedo -s data/chaste/Practical_2_3/results_*.vtu
```

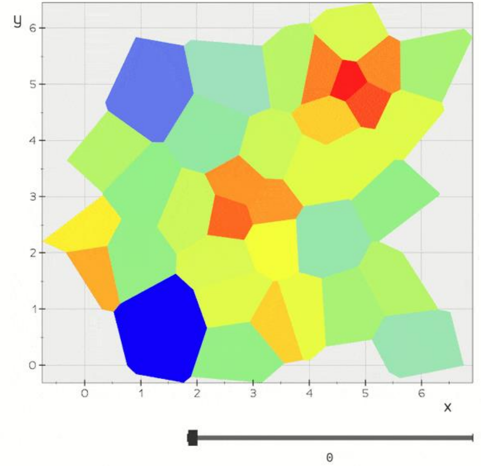
(CLI)

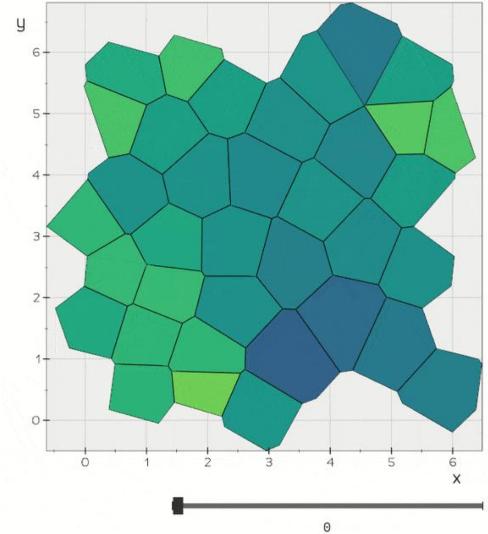
```python
from vedo.applications import Browser
Browser("data/chaste/Practical_2_3/results_*.vtu").show()
```

(script)

or:

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

ugrids = load("chaste/Practical_2_3/results_*.vtu")

meshes = []
for u in ugrids:
    m = u.alpha(1).tomesh().linewidth(1)
    m.cmap("viridis_r", vmin=0.2, vmax=2)
    meshes.append(m)

Browser(meshes).show()
```

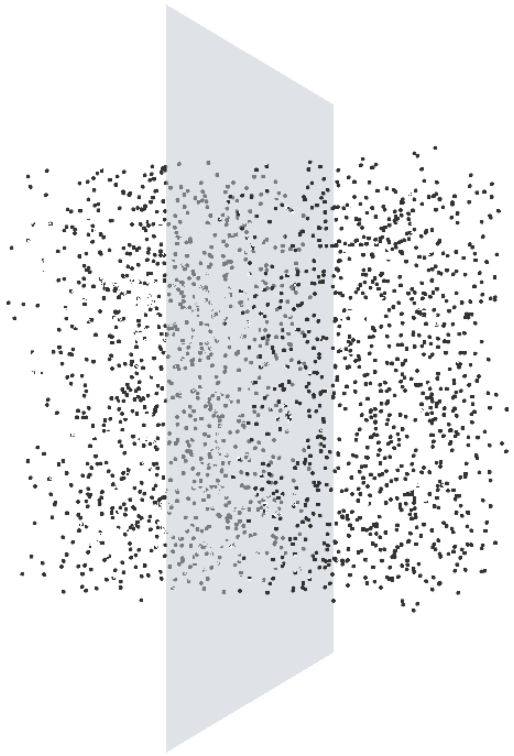




Point Clouds & Polygonal Meshes

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))
```

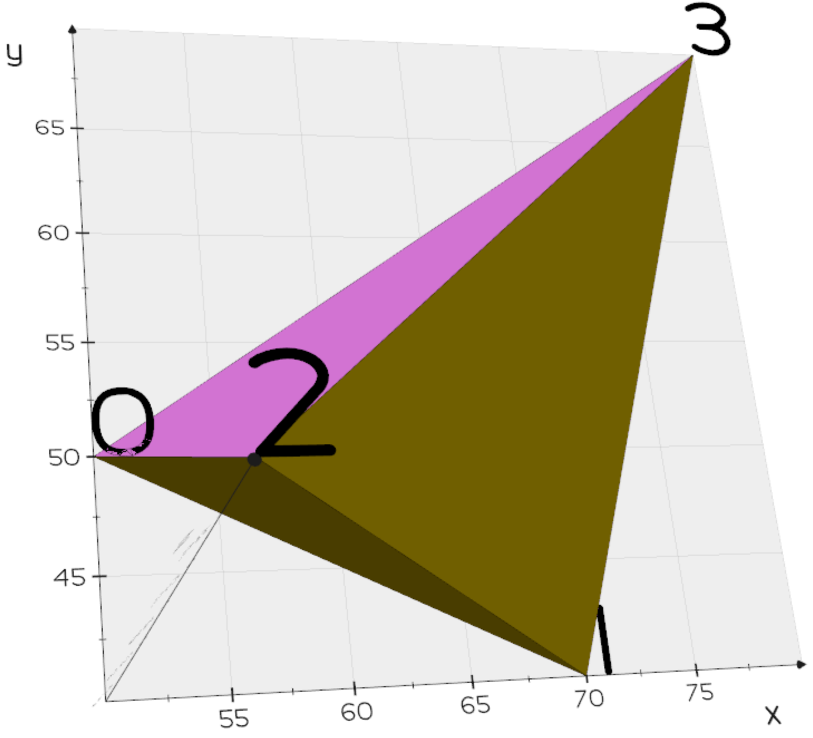
🔥 Can you create a normal distributed cloud and cut the points inside a cylinder?

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()
```

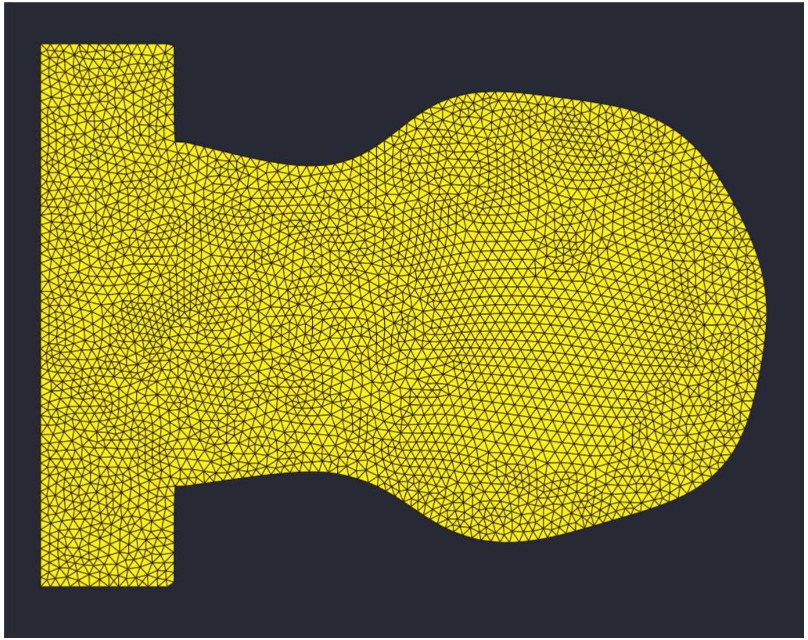
`vedo --run buildmesh`



Create a polygonal mesh

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

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



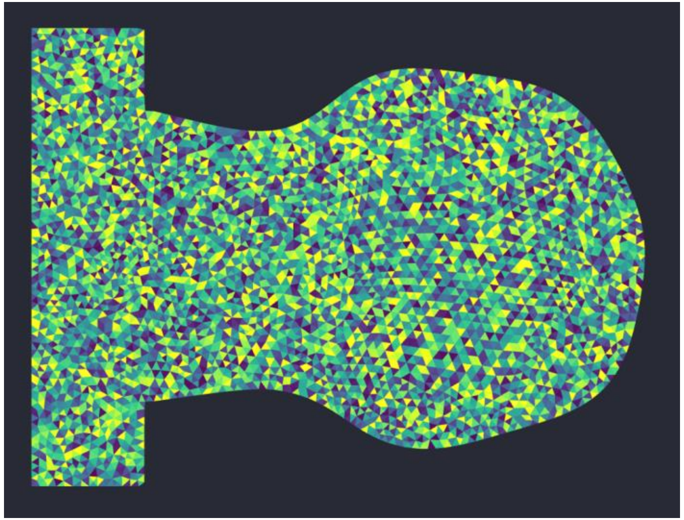
`05-gene_mesh.ipynb`

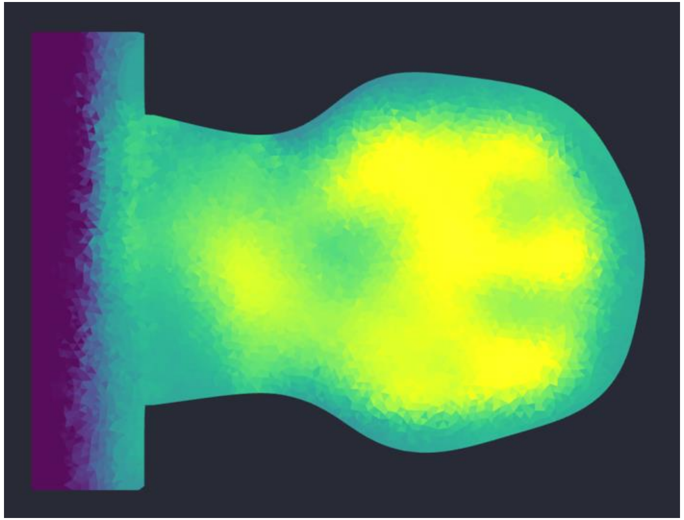
Add gene data associated with cells

```
# 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/sox9_data/gene_data.npy`



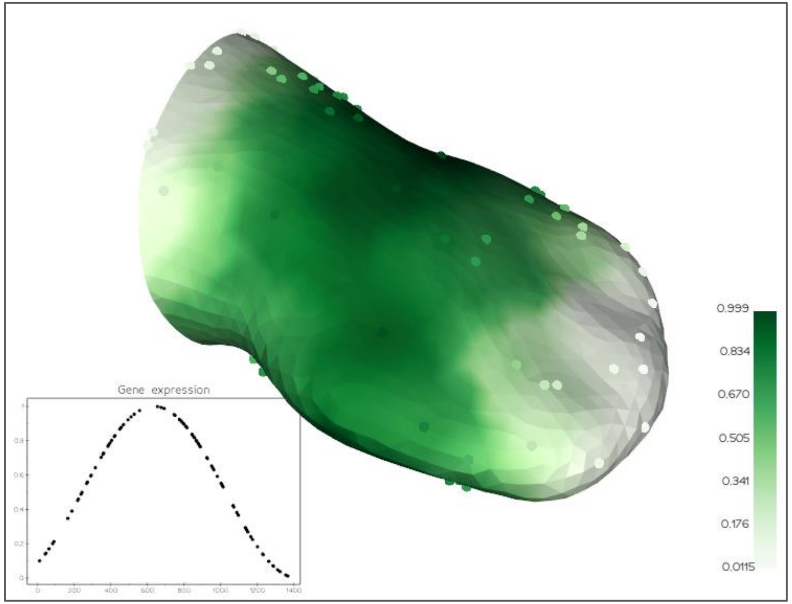


06-interpolate_scalar.ipynb

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)
```

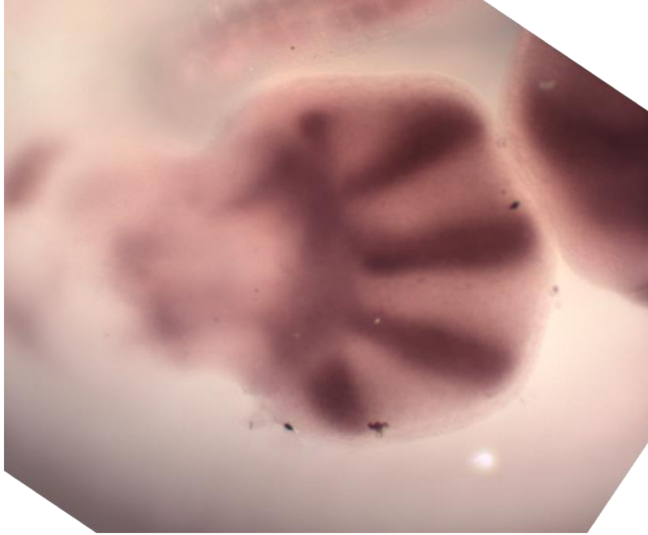


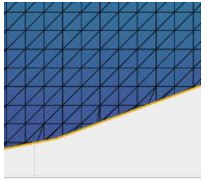
07-grab_scalars.ipynb

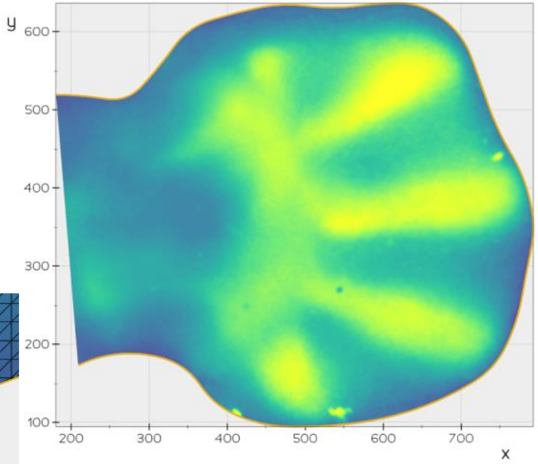
Mesh from a JPG image

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

```python
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()
```





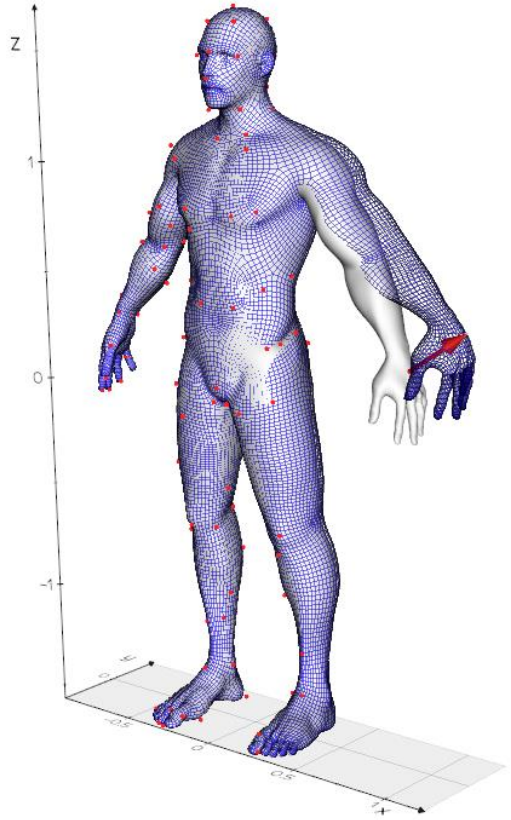


08-warp_mesh.ipynb

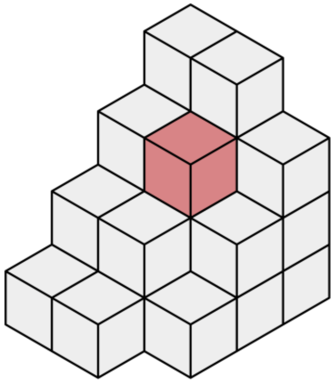
Warp a Mesh (non-linear registration)

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)
```

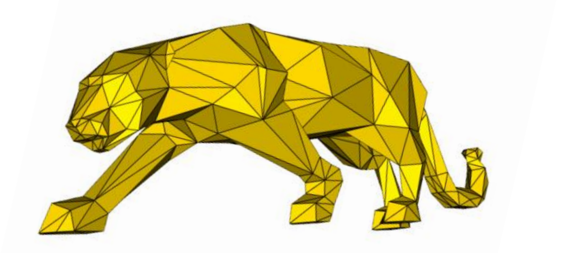


Volumes (eg TIFF stacks)

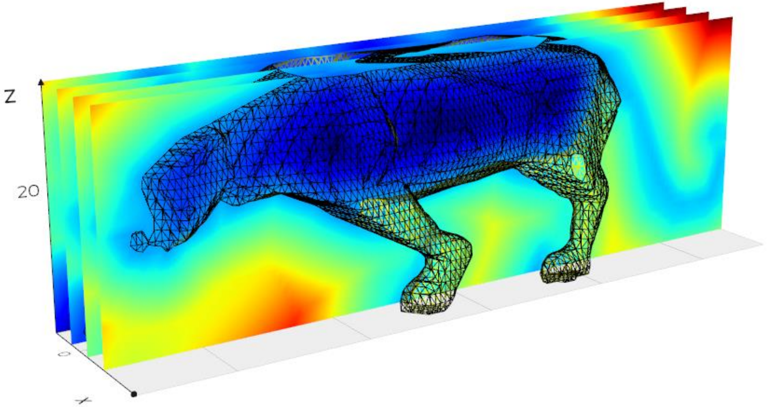


Compute distance from a mesh

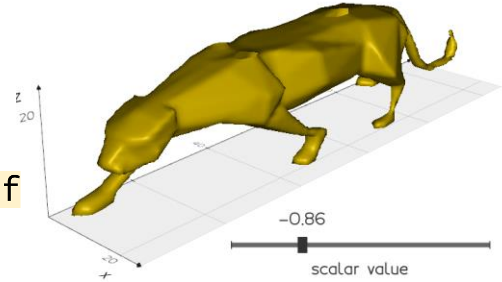
..and save it as a tiff stack



```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`



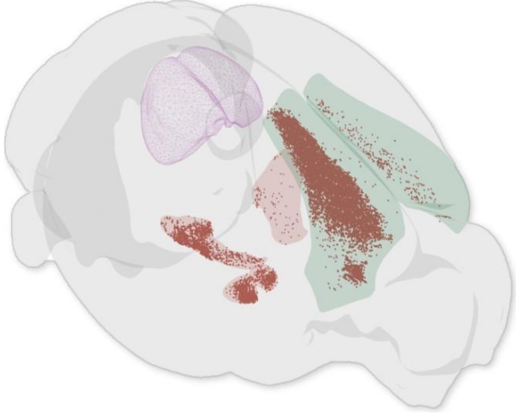
Conclusion

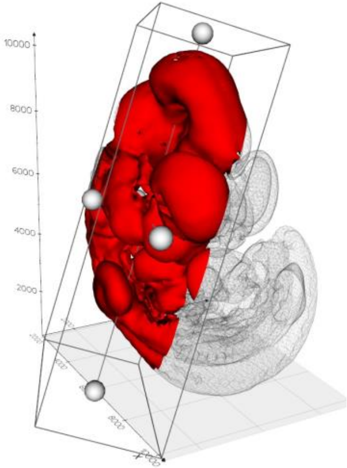
- Proved very useful in diverse applications
- Documented API with many examples
- **Happy to offer support!**

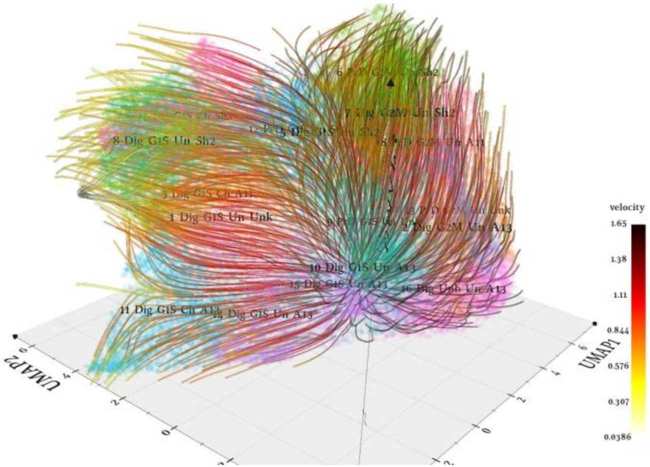
marco.musy@embl.es

laura.avino@embl.es

https://vedo.embl.es/







SHARPE LAB EMBL European Molecular Biology Laboratory

Installation links

Conda (you should have it already from Tuesday):
https://docs.conda.io/en/latest/miniconda.html

Jupyterlab:
https://jupyterlab.readthedocs.io/en/latest/getting_started/installation.html#conda

The repo:
https://github.com/LauAvinyo/vedo-embo-course/tree/main