

Basic instance segmentation

Johannes Soltwedel & Marco Musy

With materials from

Robert Haase

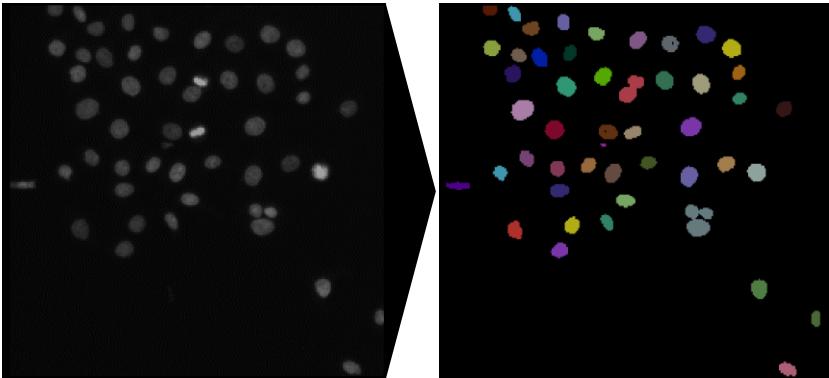
Till Korten

Aim:

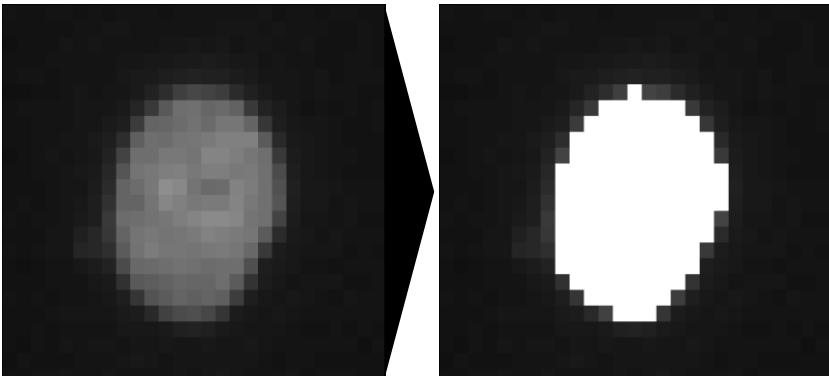
Separate background from foreground

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



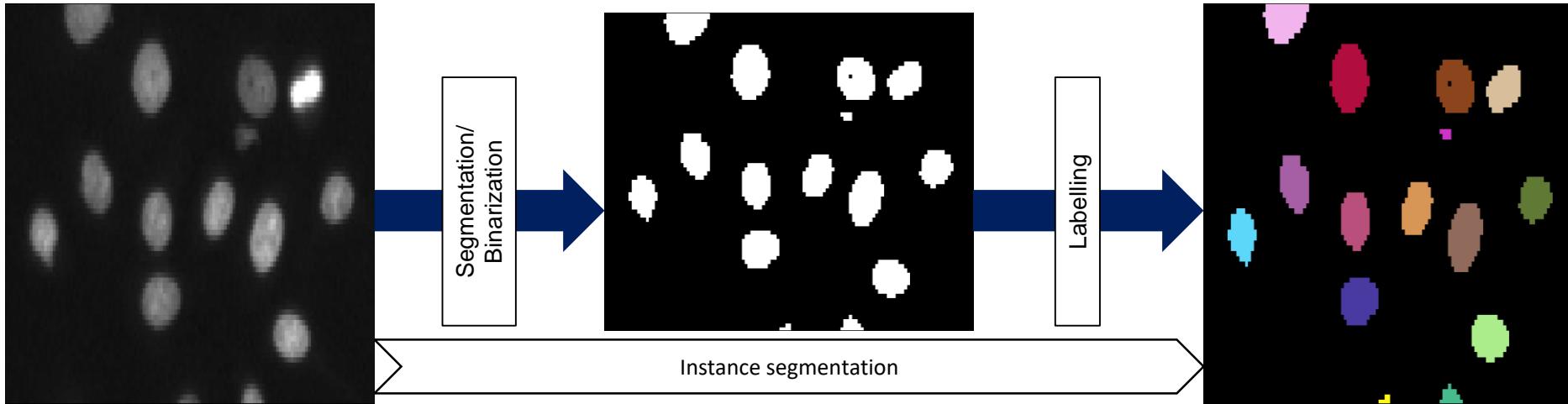
Semantic segmentation

Segmentation and labelling

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

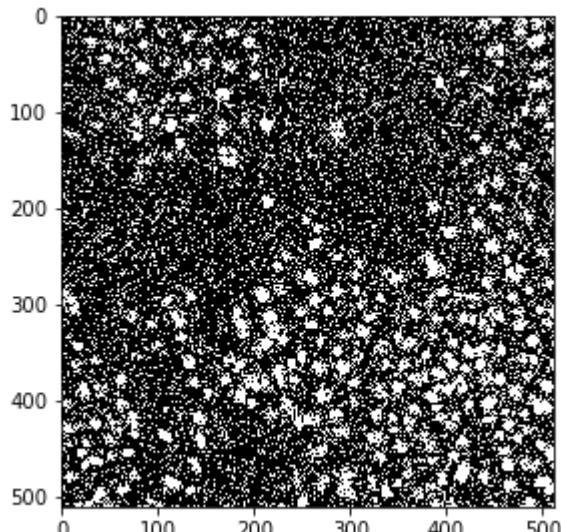
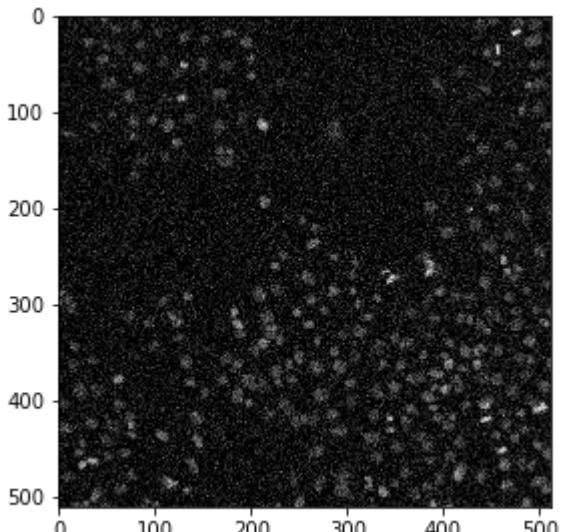
- Methods

- 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

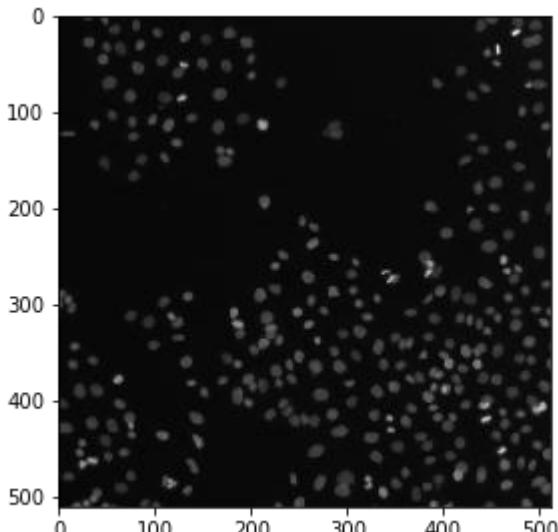


```
filtered = filters.median(image)
```

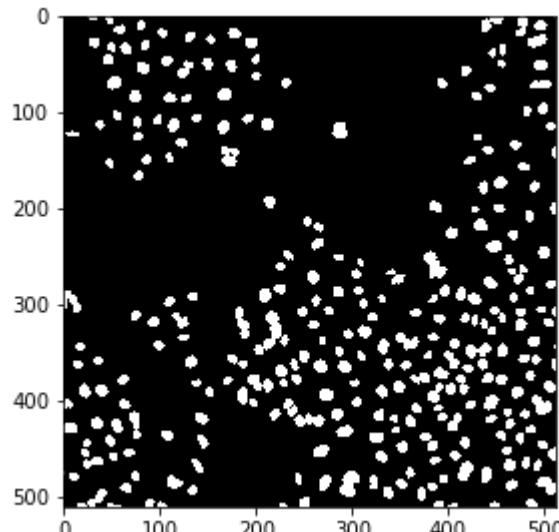
Image filtering *filters* relevant information for subsequent operations from the image!

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!

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

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

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

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

- **Statistical thresholding**: Pixels above statistical

```
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:

```
from skimage import filters
```

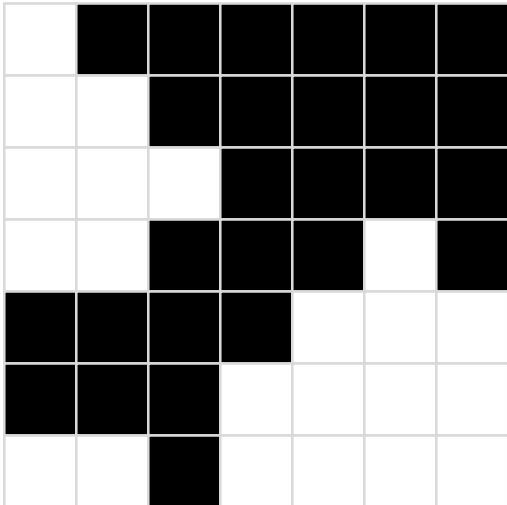
```
threshold = filters.threshold_|
```

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.

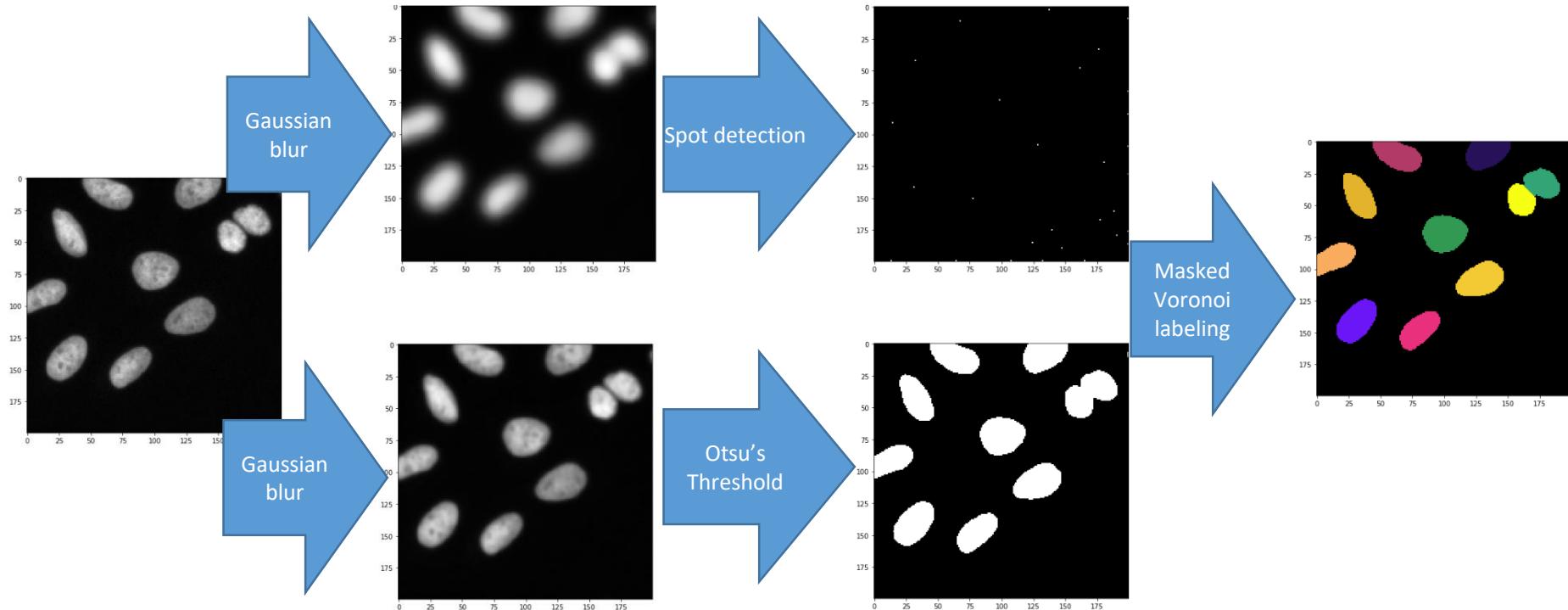


CCA

1	0	0	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0
1	1	0	0	0	3	0
0	0	0	0	3	3	3
0	0	0	3	3	3	3
2	2	0	3	3	3	3

Voronoi-Otsu-Labeling

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





Mesh processing

Johannes Soltwedel & Marco Musy

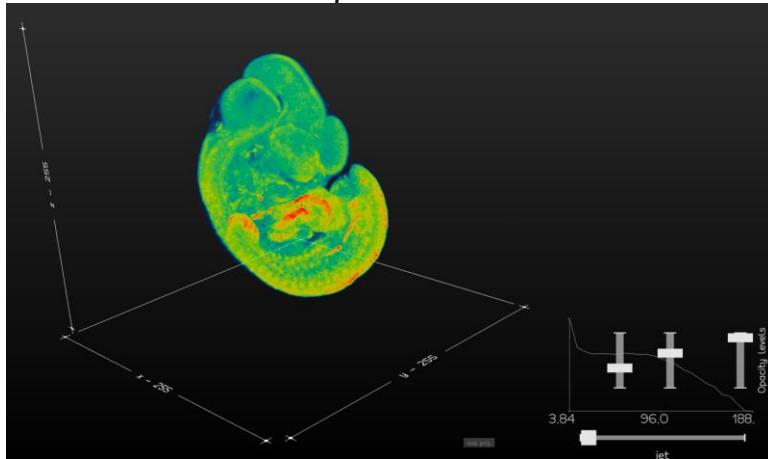
With materials from
Robert Haase
Till Korten



@jm_mightypirate

vedo

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



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])
    colorsSetColor("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)
    cylinderActorGetProperty().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')

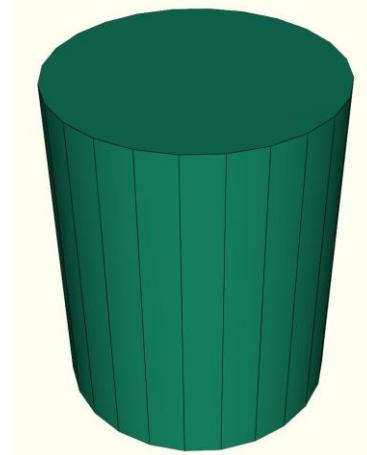
    # This allows the interactor to initialize itself. It has to be
    # called before an event loop.
    iren.Initialize()

    # 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?



TB

Preprocessing
Raw Data



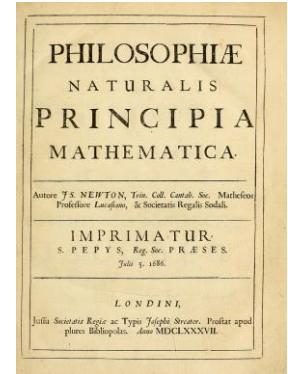
GB

v3do sits somewhere in here

Postprocessing
Data analysis



MB



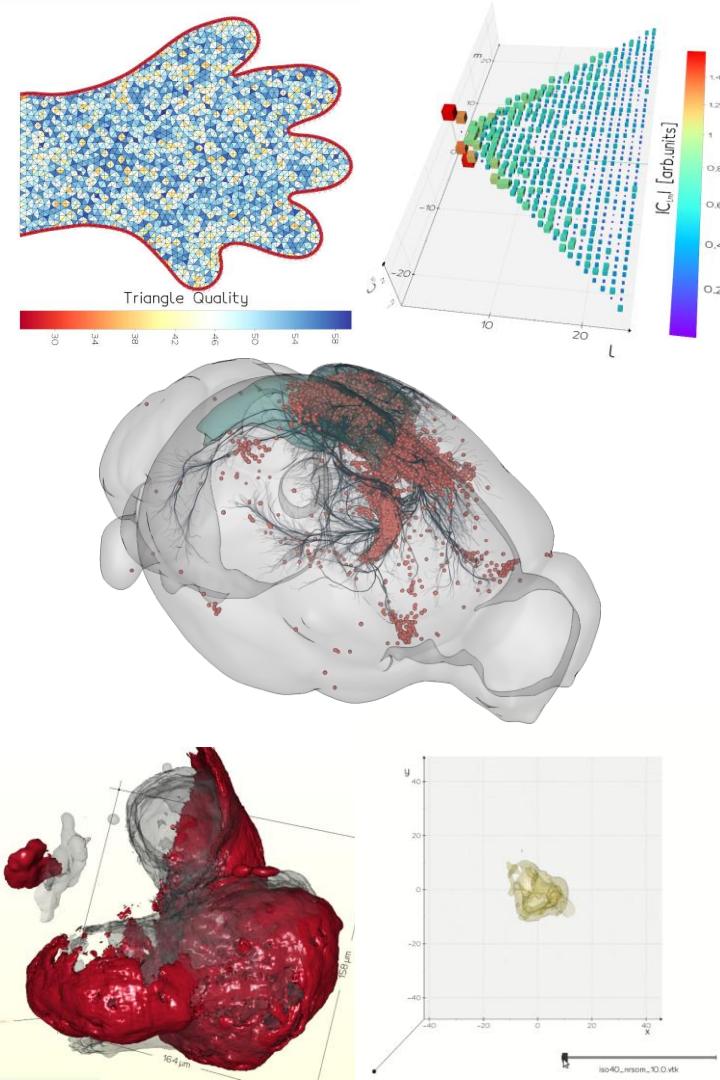
"A 3D-powered version of `matplotlib`"

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

It can prove useful with any type of data having a 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



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

Mouse hind limb

layer list



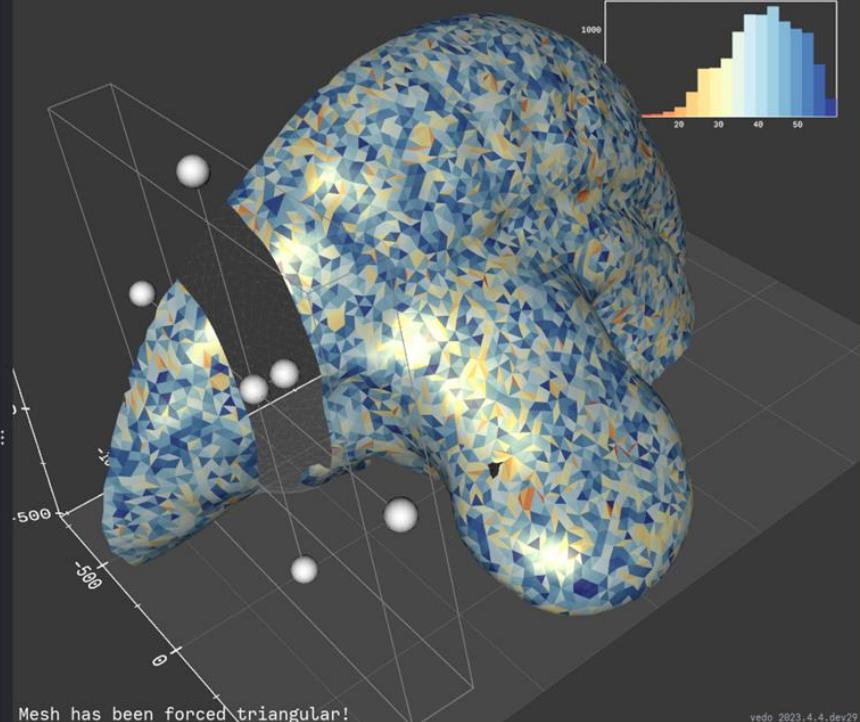
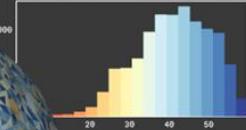
270_flank.vtk *



napari

Mesh cutter (napari-vedo-bridge)

Quality of Polygonal Faces



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

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

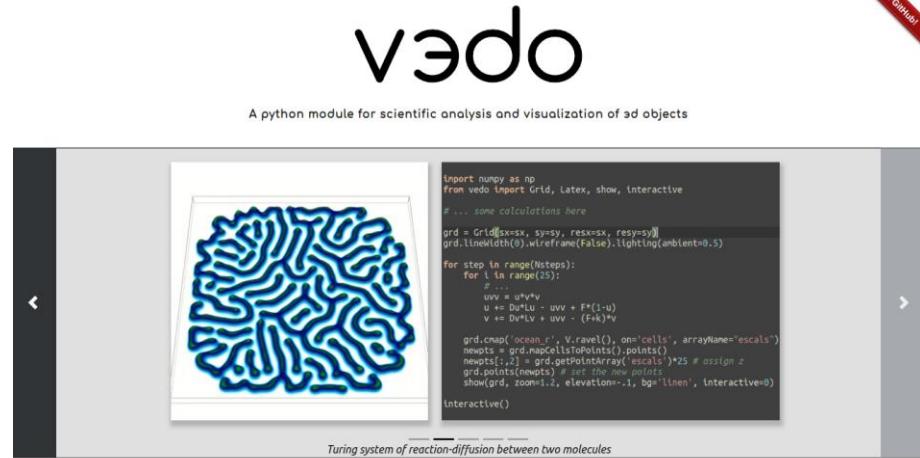
- activity

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 exemplename**



Basic Advanced Volumetric Animations Plotting Others

Search example *e.g. "mesh"* Hover mouse to see a description

3
indigo5 indigo6 indigo7
paper Woods gnu
Accent Blues BG-BG BuBG BuGN BuPu CH3map X2 GrBu Greens
afmhot binary bone cool

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



Vedo - FOREVER [LIVE SESSION]

 Vedo 
1.02M subscribers

Subscribe

 9.3K   Share 

vedo

API Documentation

Search...

Contents

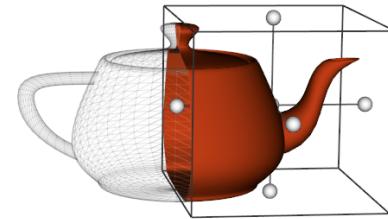
- Install and Test
- Command Line Interface
- Export your 3D scene to file
- File format conversion
- Running in a Jupyter Notebook
- Running on a Server
- Running in a Docker container
- Generate a single executable file
- Getting help

Submodules

- addons
- applications
- assembly
- base
- colors

vedo

license: MIT | Anaconda.org: 2023.4.5 | Ubuntu 23.04 package: 2023.4.3 | DOI: 10.5281/zenodo.5842090



A python module for scientific analysis of 3D objects and point clouds based on [VTK](#) and [numpy](#).

Check out the [GitHub repository here](#).

Install and Test

```
pip install vedo  
# Or, install the latest development version:  
pip install -U git+https://github.com/marcomusy/vedo.git
```

Then

```
import vedo  
vedo.Cone().show(axes=1).close()
```

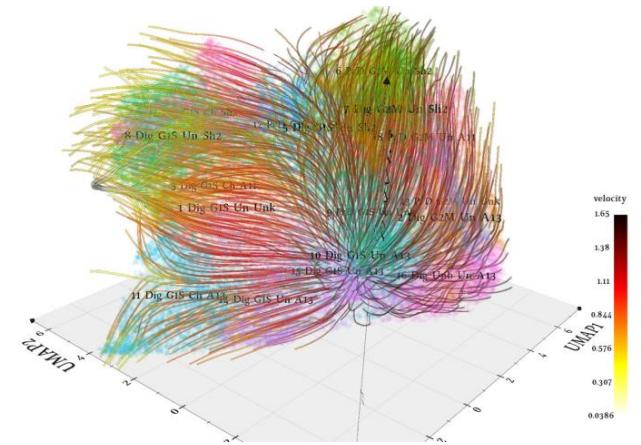
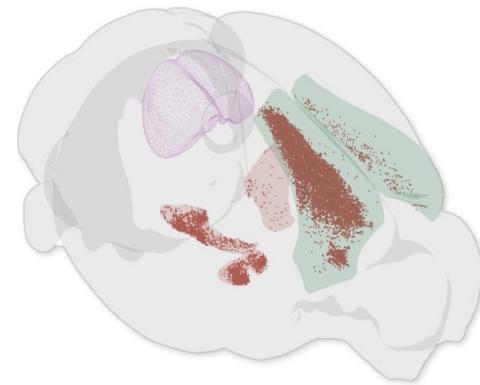
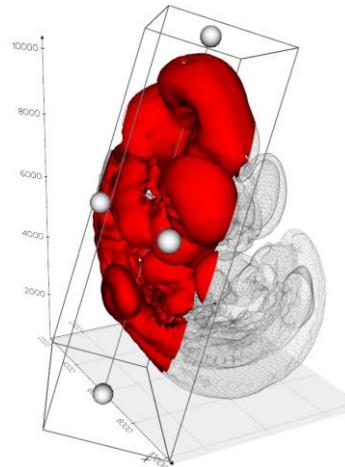
Conclusion

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

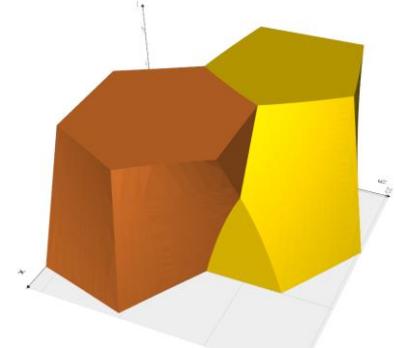
marco.musy@embl.es

laura.avino@embl.es

<https://vedo.embl.es/>



vedo practicals



Installing steps

1. Get the repo.

Opening a terminal (anaconda).

```
> git clone https://github.com/LauAvinyo/vedo-embo-course.git
> cd vedo-embo-course
```

2. Install dependencies:

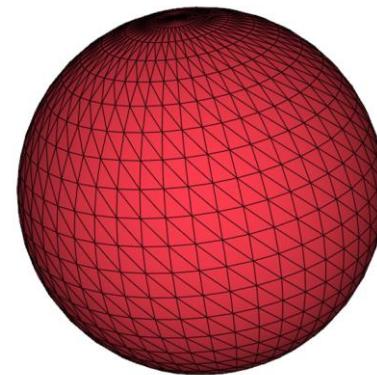
```
> mamba install vedo
```

3. For Johannes: Doublecheck whether vedo is in devbio-napari

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

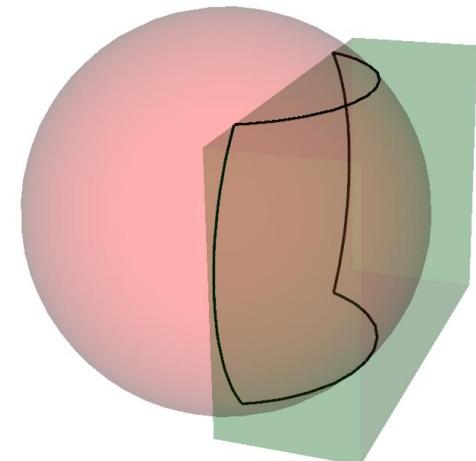


Press "h" in
rendering window

```
# 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

Example from Tuesday Chaste tutorial

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

(CLI)

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

(script)

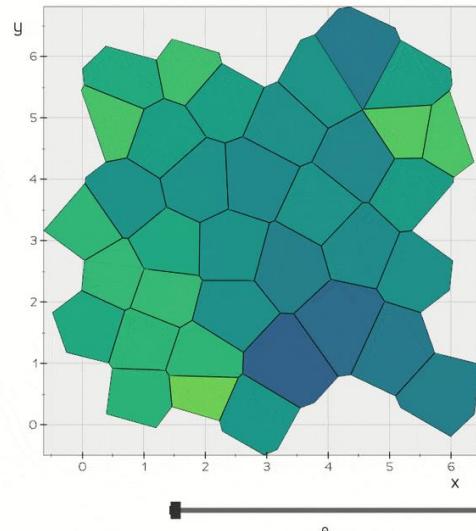
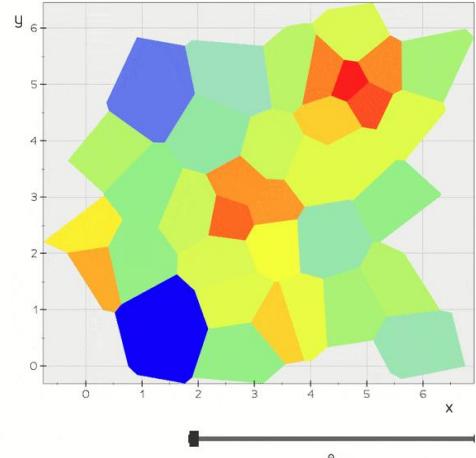
Or:

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

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

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

Browser(meshes).show()
```



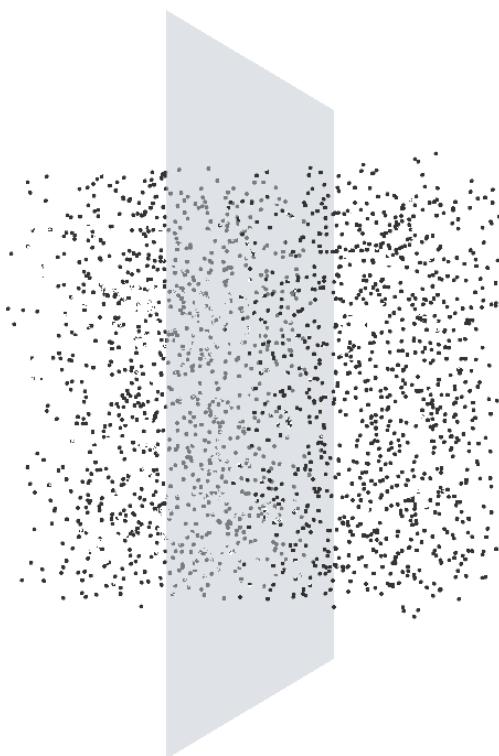
Point Clouds & Polygonal Meshes

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

```
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)]

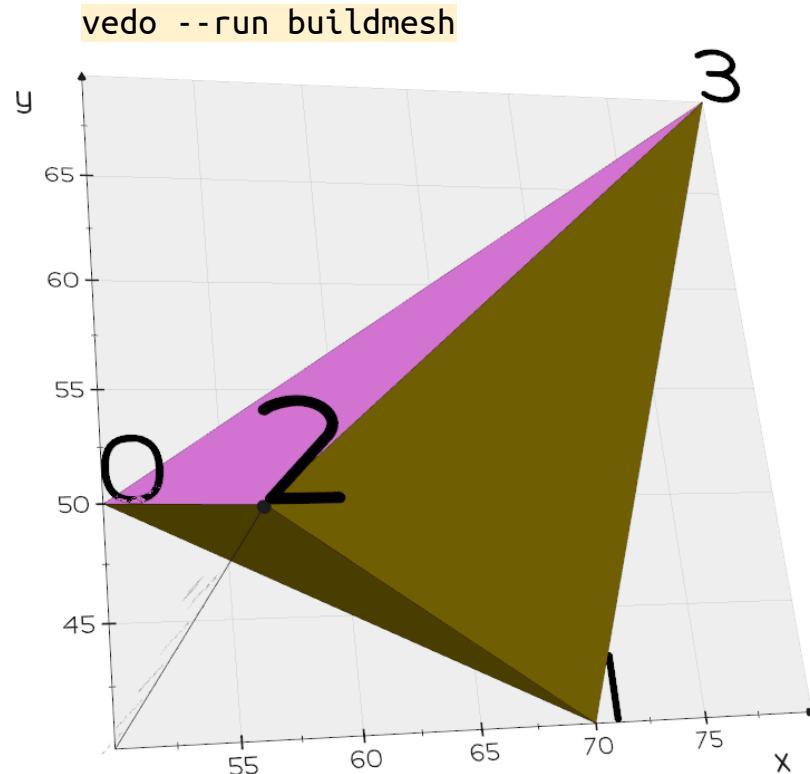
# Build the polygonal Mesh object from the vertices and faces
mesh = Mesh(verts, faces)

# 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())

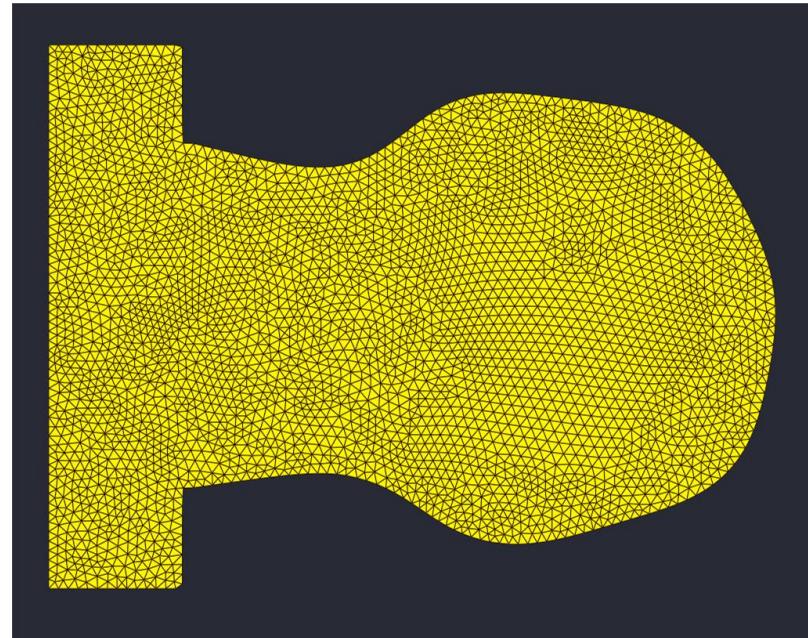
# Show the mesh, vertex labels, and docstring
show(mesh, labs, viewup='z', axes=1).close()
```



Create a polygonal mesh

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

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

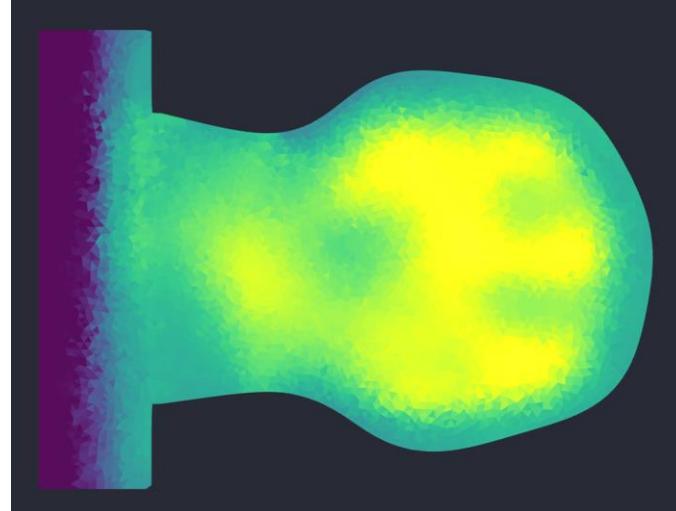
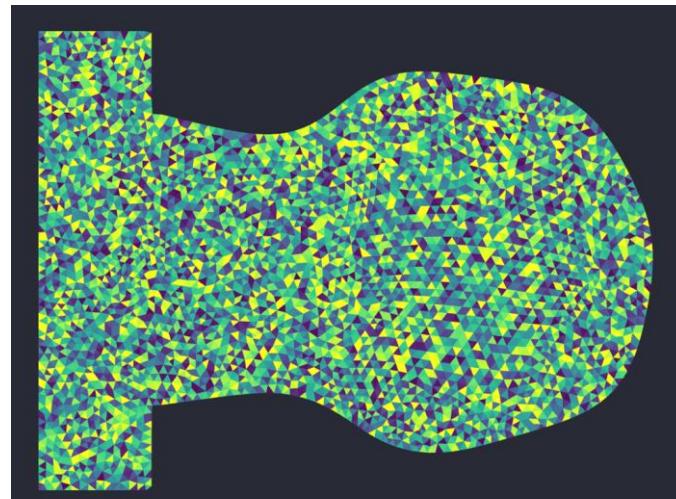


Add gene data associated with cells

```
# 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?

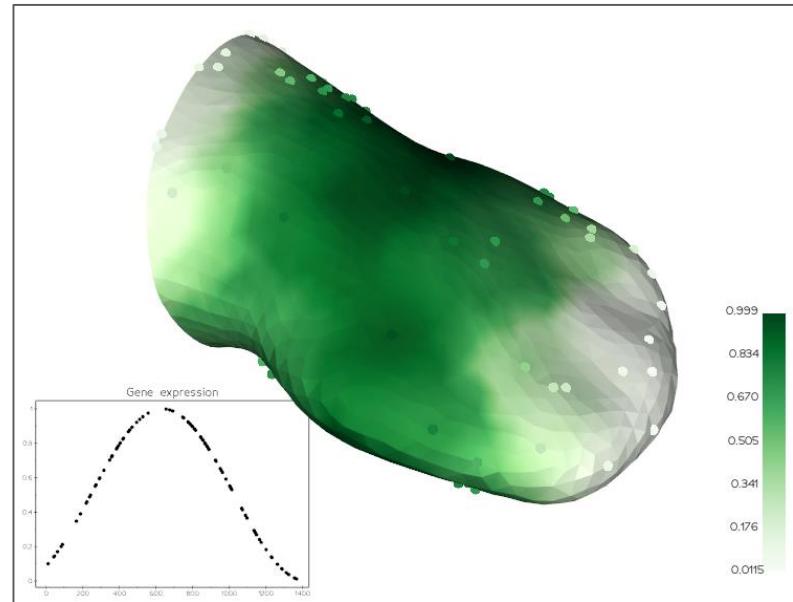
Use `./data/sox9_data/gene_data.npy`



Interpolate data from sparse measurements in 3D

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

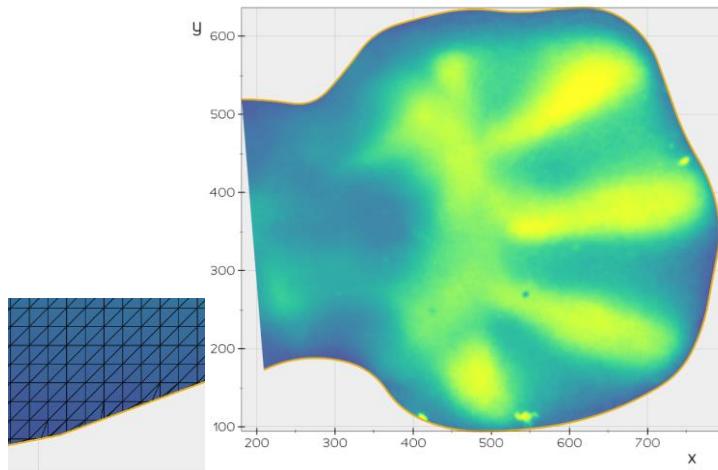
```
1 import numpy as np
2 from vedo import dataurl, Mesh, Points, show
3 from vedo.pyplot import plot
4
5 # Load a mesh of a mouse limb at 12 days of development
6 msh = Mesh(dataurl + "290.vtk")
7
8 # Pick 100 points where we measure the value of a gene expression
9 ids = np.random.randint(0, msh.npoints, 100)
10 pts = msh.points()[ids]      # slice the numpy array
11 x = pts[:, 0]                # x coordinates of the points
12 gene = np.sin((x+150)/500)**2 # we are making this up!
13
14 # Create a set of points with those values
15 points = Points(pts, r=10).cmap("Greens", gene)
16
17 # Interpolate the gene data onto the mesh, by averaging the 5 closest points
18 msh.interpolate_data_from(points, n=5).cmap("Greens").add_scalarbar()
19
20 # Create a graph of the gene expression as function of x-position
21 gene_plot = plot(x, gene, lw=0, title="Gene expression").as2d(scale=0.5)
22
23 # Show the mesh, the points and the graph
24 show(msh, points, gene_plot)
25
```



Mesh from a JPG image

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

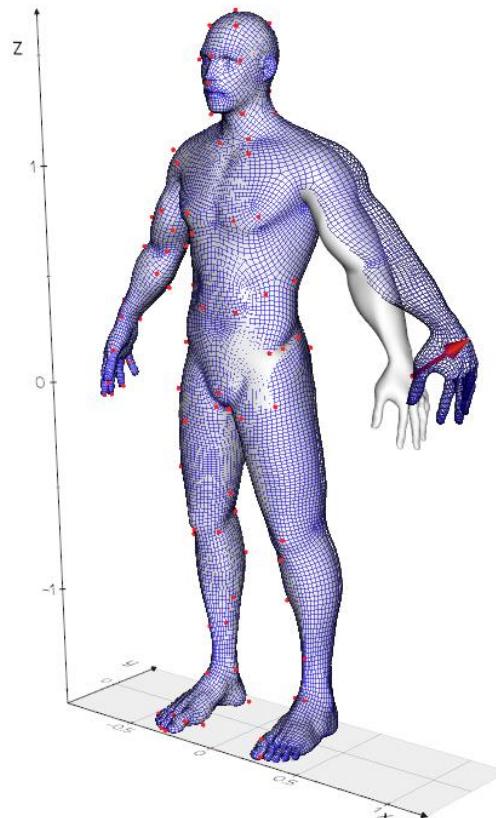
```
1  from vedo import Picture, settings, show
2  from vedo.applications import SplinePlotter
3
4  settings.default_backend = "vtk"
5
6  pic = Picture("data/sox9_exp.jpg").bw()
7
8  plt = SplinePlotter(pic)
9  plt.show(mode="image", zoom='tight')
10 outline = plt.line
11 plt.close()
12
13 print("Cutting with outline...")
14 msh = pic.tomesh().triangulate().cmap("viridis_r")
15 cut_msh = msh.clone().cut_with_point_loop(outline)
16
17 cut_msh.interpolate_data_from(msh, n=3)
18 show(cut_msh, outline, axes=1).close()
```



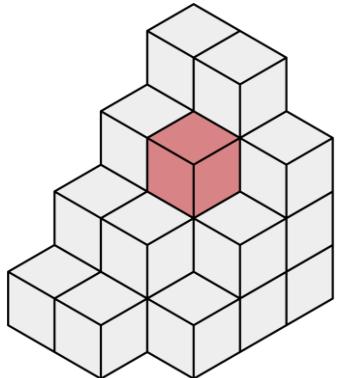
Warp a Mesh (non-linear registration)

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

```
4  from vedo import dataurl, Mesh, Arrows, show
5
6  # Load a mesh
7  mesh = Mesh(dataurl+"man.vtk").color("white")
8
9  # Create a heavily decimated copy with about 200 points
10 # (to speed up the computation)
11 mesh_dec = mesh.clone().triangulate().decimate(n=200)
12
13 sources = [[0.9, 0.0, 0.2]] # this point moves
14 targets = [[1.2, 0.0, 0.4]] # ...to this.
15 for pt in mesh_dec.points():
16     if pt[0] < 0.3:          # while these pts don't move
17         sources.append(pt)   # (e.i. source = target)
18         targets.append(pt)
19
20 # Create the arrows representing the displacement
21 arrow = Arrows(sources, targets)
22
23 # Warp the mesh
24 mesh_warped = mesh.clone().warp(sources, targets)
25 mesh_warped.c("blue").wireframe()
26
27 # Show the meshes and the arrow
28 show(mesh, mesh_warped, arrow, axes=1)
```



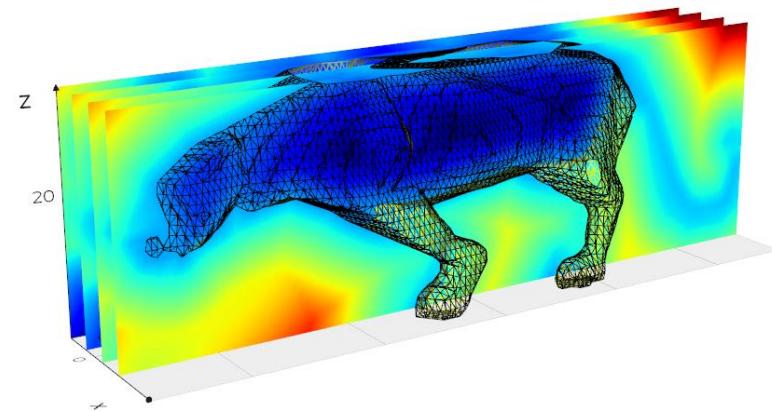
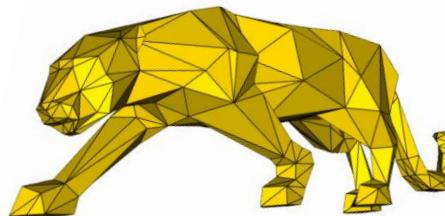
Volumes (eg TIFF stacks)



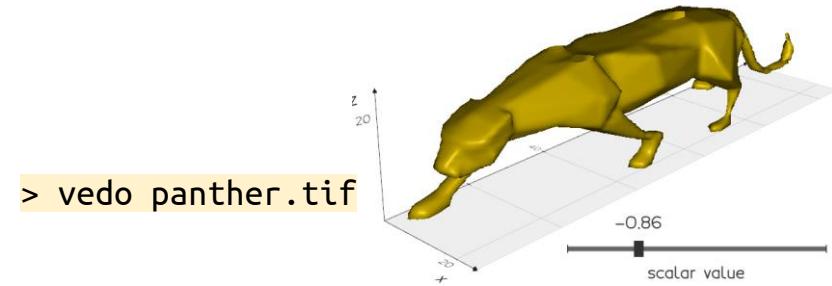
Compute distance from a mesh

..and save it as a tiff stack

```
1  from vedo import *
2
3  msh = Mesh(dataurl + "panther.stl")
4
5  vol = msh.signed_distance(dims=[25,125,25])
6  iso = vol.isosurface(0.0)
7
8  plt = Plotter()
9  plt += iso.wireframe()
10
11 for i in range(0, 25, 5):
12     plt += vol.xslice(i).cmap("jet")
13
14 vol.write("panther.tif")
15 plt.show(axes=1)
```



> vedo panther.tif

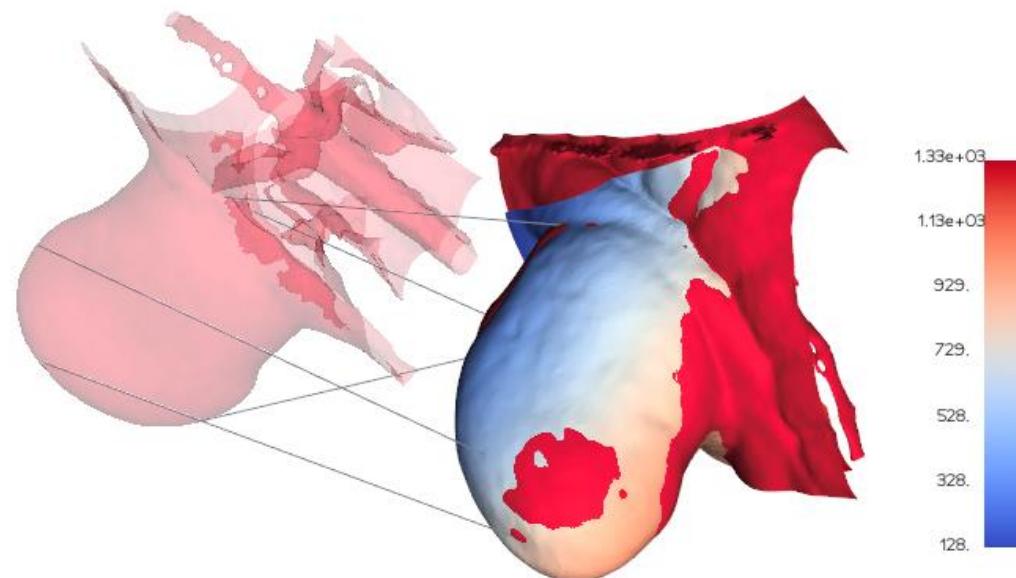


Extras: Linear vs Non-Linear Registration

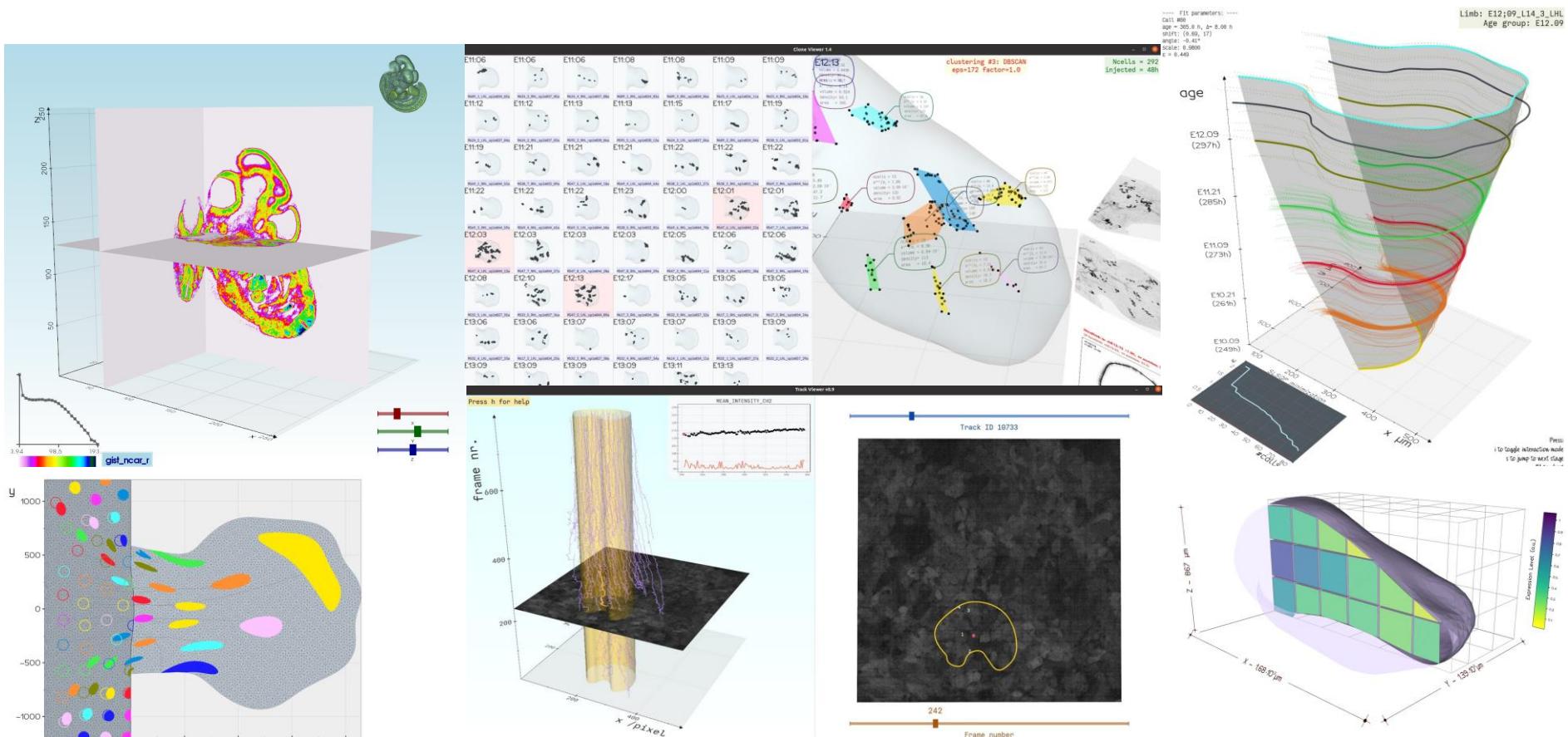
Naive registration (ICP) might fail!

Try to:

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



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



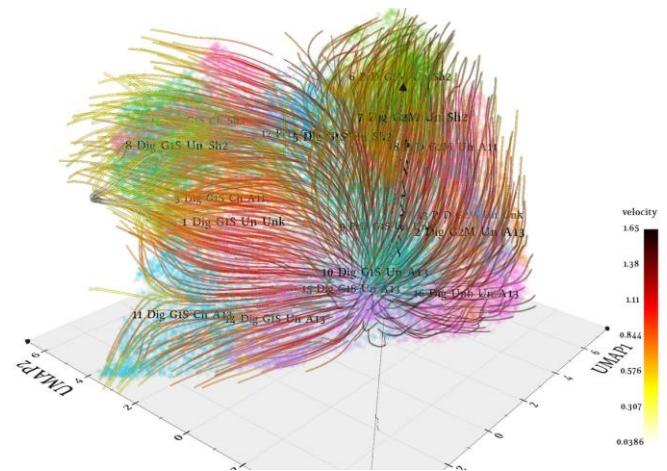
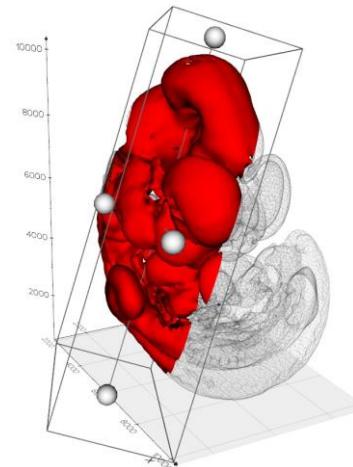
Conclusion

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

marco.musy@embl.es

laura.avino@embl.es

<https://vedo.embl.es/>



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>