Intro
ooooooo

Installing TTK
ooo

Workflows
oooooooooooo

Cluster demo
oo

Summary
o

The Topology ToolKit

ALEX RAZOUMOV
alex.razoumov@westgrid.ca

Intro
ooooooo

Installing TTK
ooo

Workflows
oooooooooooo

Cluster demo
oo

Summary
o

Zoom controls

- Please mute your microphone and camera unless you have a question

- To ask questions at any time, type in Chat, or Unmute to ask via audio

- Raise your hand in Participants

- This talk is being recorded
  - your name might appear in the recording (if you want, you can change it)
  - video will be posted at https://westgrid.github.io/trainingMaterials under one of the top-menu topics

- Email training@westgrid.ca

Intro
●○○○○○○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○○

Cluster demo
○○

Summary
○

Today's topic

- Topological data analysis (TDA) can be very useful in analyzing complex multi-dimensional datasets – examples in the next slide
  - filter out features by their persistence value
  - domain segmentation
  - topological simplification

- I have zero background in topology / TDA

$\Rightarrow$ Let's approach this from a non-topologist / research scientist perspective

$\Rightarrow$ Our today's goal is not to learn topology, but to look at an open-source TDA tool that is integrated with VTK and ParaView

Topology-based analysis
This slide is based on Attila Gyulassy's introductory topology tutorial (U. of Utah)

- Measurable topological attributes:
  - measurement of connectedness: pick any two points + find an inside path to connect them
  - count non-contractable cycles: pick a closed loop inside that you can't squeeze down to a point
  - and many others

- Applications in:
  - molecular analysis: e.g. pick up atomic bonds, atoms from the 3D electronic probability density
  - materials analysis: porosity measurement, battery design, defect identification, cavity deformation
  - neural pathways (connectomics)
  - CFD: turbulence and vorticity, buble formation rate, ocean eddies
  - combustion analysis: ignition kernels
  - geological features
  - image segmentation

- Phenomenon of interest in a scalar field in the data $\Rightarrow$ derive its measurable topological equivalent or abstraction

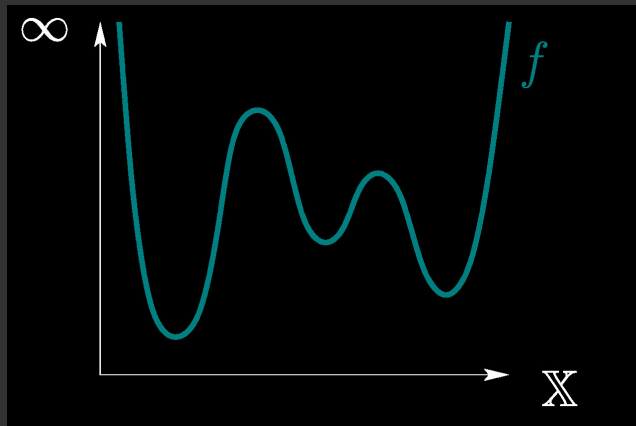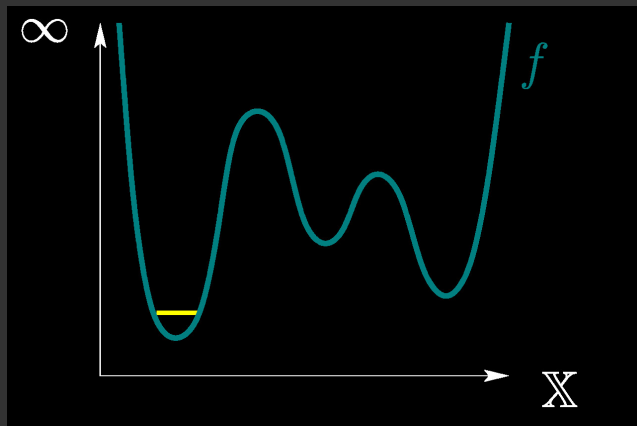## Persistence intervals of a 1D scalar function

Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Evolution of the topology of connected components during a filtration from $-\infty$ to $+\infty$
- Pair thresholds (critial points) that create components with those that destroy them
- Component persistence = difference in function value between component's birth and death

Intro
○○○●○○○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○

Cluster demo
○○

Summary
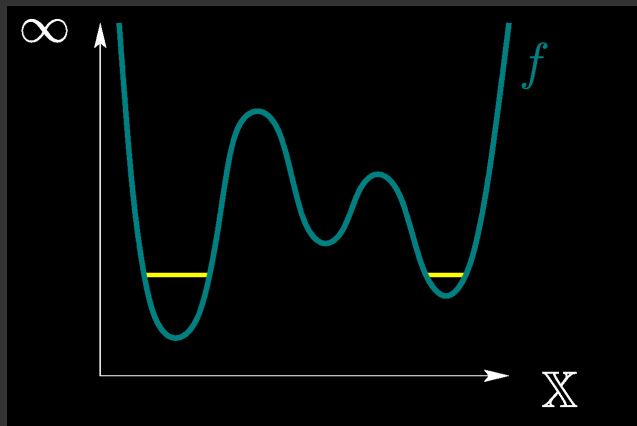○

Persistence intervals of a 1D scalar function

Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Evolution of the topology of connected components during a filtration from $-\infty$ to $+\infty$
- Pair thresholds (critial points) that create components with those that destroy them
- Component persistence = difference in function value between component's birth and death

Intro
ooo●ooo
Installing TTK
ooo
Workflows
oooooooooooo
Cluster demo
oo
Summary
o

Persistence intervals of a 1D scalar function

Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Evolution of the topology of connected components during a filtration from $-\infty$ to $+\infty$
- Pair thresholds (critial points) that create components with those that destroy them
- Component persistence = difference in function value between component's birth and death

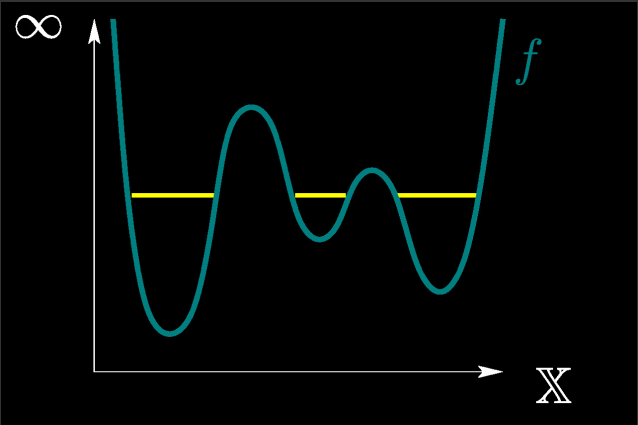# Persistence intervals of a 1D scalar function

Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Evolution of the topology of connected components during a filtration from $-\infty$ to $+\infty$
- Pair thresholds (critial points) that create components with those that destroy them
- Component persistence = difference in function value between component's birth and death

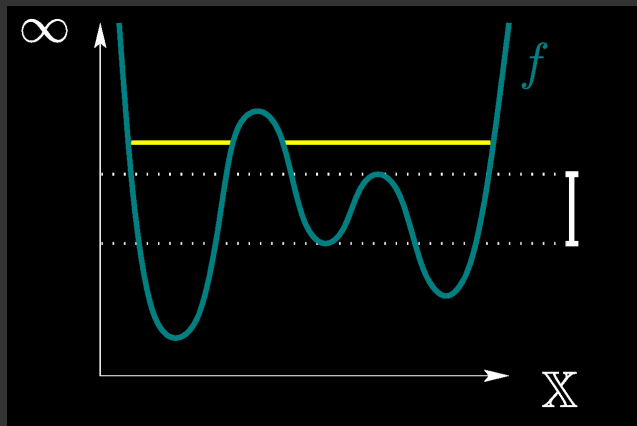# Persistence intervals of a 1D scalar function

Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Evolution of the topology of connected components during a filtration from $-\infty$ to $+\infty$
- Pair thresholds (critial points) that create components with those that destroy them
- Component persistence = difference in function value between component's birth and death

Intro
○○●○○○○
Installing TTK
○○○
Workflows
○○○○○○○○○○○○
Cluster demo
○○
Summary
○
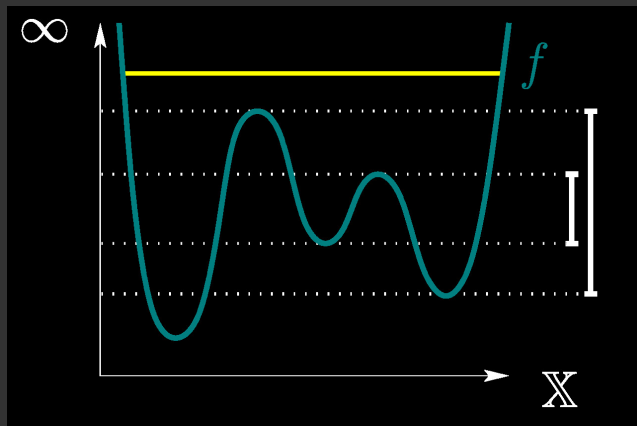
Persistence intervals of a 1D scalar function

Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Evolution of the topology of connected components during a filtration from $-\infty$ to $+\infty$
- Pair thresholds (critial points) that create components with those that destroy them
- Component persistence = difference in function value between component's birth and death

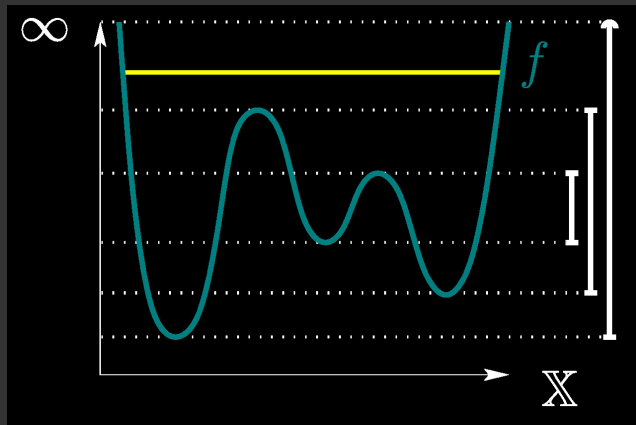## Persistence intervals of a 1D scalar function

Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Evolution of the topology of connected components during a filtration from $-\infty$ to $+\infty$
- Pair thresholds (critial points) that create components with those that destroy them
- Component persistence = difference in function value between component's birth and death

Intro
○○○●○○○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○

Cluster demo
○○

Summary
○

Persistence diagram

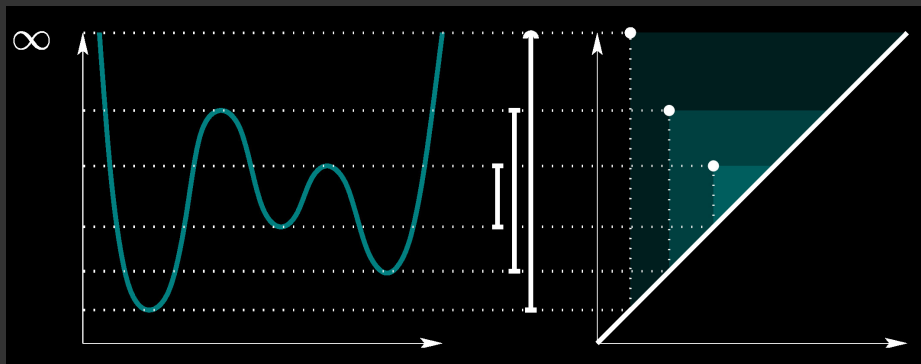
Figure copied from David Cohen-Steiner's slides on Topological Persistence

- Persistence diagram: x = function value at feature's birth, y = function value at feature's death
- Small features are mapped closer to the diagonal
- Critical points in 1D: minima and maxima
- Domain segmentation (monotone regions) between critical points, based on the gradient sign
- Mountains = monotone pieces around maxima, basins = monotone pieces around minima

Intro
○○○○●○○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○

Cluster demo
○○

Summary
○
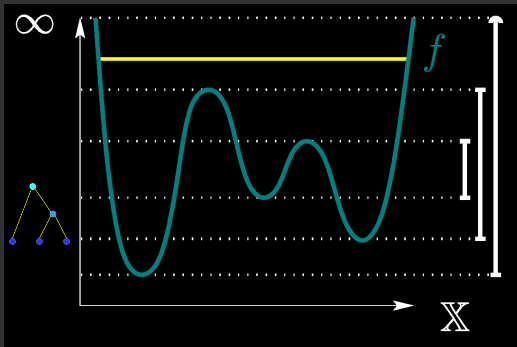
- Join tree of connected components
- Leaves correspond to minima
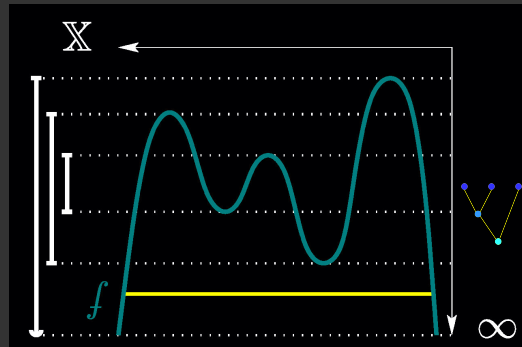
- Split tree of connected components
- Leaves correspond to maxima

In TTK you can easily:

1. compute persistence diagrams and persistence curves
2. apply a filter to build a tree; areas attached to the leaves have RegionType = 1
3. perform topological segmentation: subdivide domain into (e.g.) monotone regions
4. perform topological simplification: delete some leaves, typically below some threshold persistence

Intro
○○○○○○●○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○

Cluster demo
○○

Summary
○

Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines

Intro
ooooooooo

Installing TTK
ooo

Workflows
oooooooooooo

Cluster demo
oo

Summary
o

Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines

Intro
○○○○○○●○

Installing TTK
○○○

Workflows
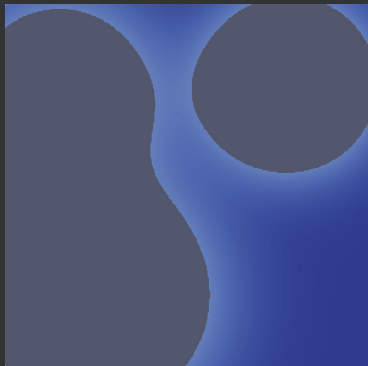○○○○○○○○○○○○

Cluster demo
○○

Summary
○

Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines

Intro
○○○○○○●○

Installing TTK
○○○

Workflows
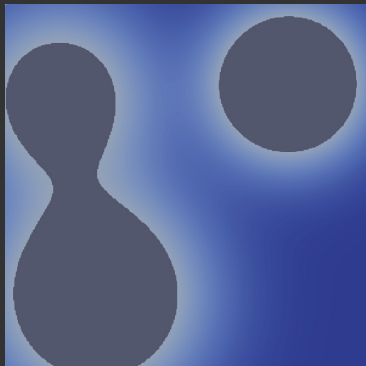○○○○○○○○○○○○

Cluster demo
○○

Summary
○

Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines
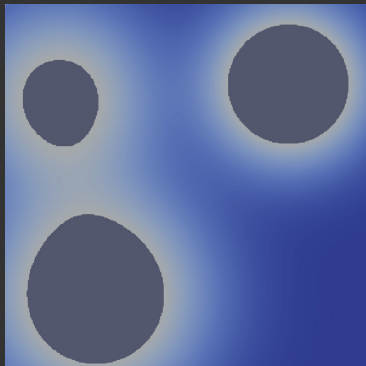
Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines

Intro
○○○○○○●○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○

Cluster demo
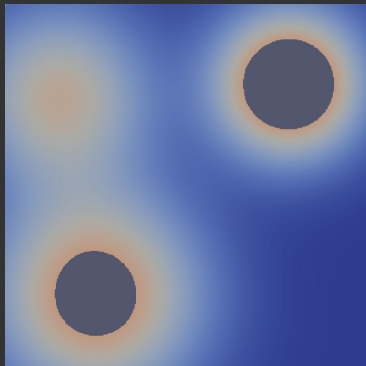○○

Summary
○

Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines
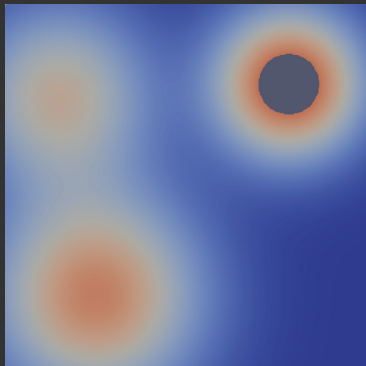
Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines

Intro
ooooooeo

Installing TTK
ooo

Workflows
oooooooooooooo

Cluster demo
oo

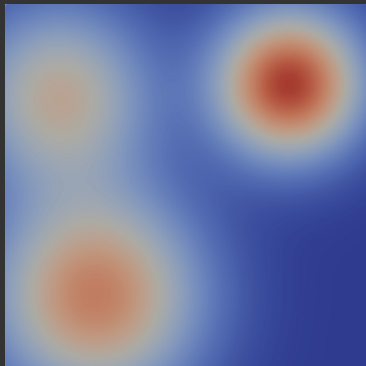Summary
o

Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines

Intro
○○○○○○●○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○

Cluster demo
○○

Summary
○

Scalar function in 2D

- Sweep in function value from $-\infty$ to $+\infty$

- Coloured regions shows the subdomain with function value lower than the sweep value

- This time single component with multiple holes that
  - split at the saddles
  - disappear at the maxima

- Critical points in 2D: minima, maxima, and saddle points

- Domain segmentation (monotone regions) between integral lines (orthogonal to the contours), based on the gradient sign

- Same definition for mountains and basins, but there are also ridge lines and valley lines

Scalar function in 3D

- The topology becomes more complex
- 4 types of critical points: minima, maxima, and two kinds of saddles
  - 1-saddles are located between 2 minima, 2-saddles are located between 2 maxima
- Domain segmentation (monotone regions) between integral surfaces, based on the gradient sign
- Monotone regions combined into mountains and basins, with ridge lines and valley lines and saddle connector lines, ridge surfaces (separating the mountains) and valley surfaces (separating the basins)

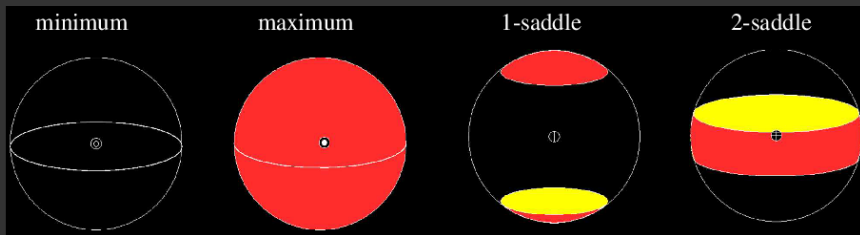| minimum | maximum | 1-saddle | 2-saddle |

Figure copied from Guoning Chen's slides on Morse-Smale Complex

Intro
0000000

Installing TTK
●oo

Workflows
000000000000

Cluster demo
oo

Summary
o

TTK = the Topology ToolKit

- Toplogical analysis of multi-dimensional scalar functions

- https://topology-tool-kit.github.io
  - great documentation, many tutorials and step-by-step videos

- Open source, development since 2014, first public release in 2017

- Lead author Julien Tierny + active development community

- Some workflows in this presentation were taken from the excellent half-day TTK tutorial at IEEE VIS 2020

- Interfaces:
  - pure C++
  - VTK/C++ (~3X shorter code)
  - ParaView Python (~5X shorter code) and ParaView plugin

Installing TTK

Compilation instructions `https://topology-tool-kit.github.io/installation.html`

Here is what I did on my Macbook:

1. Create Python 3.7 (older than my default!) environment and install scikit-learn and vtk==8.2.0
2. Install latest Qt with Homebrew into `/usr/local`
3. Download/copy Intel's Threading Building Blocks library to `/opt/local/{lib,include}`
4. Download ParaView 5.6.1 and TTK 0.9.8
5. Apply TTK patch to ParaView's source code (only available up to 5.6)
6. Compile ParaView with TBB and Qt5
    - ParaView compile scripts were selecting the default system's Qt $\Rightarrow$ ParaView was hanging on startup
    - had to specify `-DQt5_DIR=/usr/local/Cellar/qt/5.15.1/lib/cmake/Qt5`
7. `export PV_PLUGIN_PATH=/usr/local/lib/plugins`
8. Compile TTK (showing the difficult flags only)

```
-DParaView_DIR=~/ttk/ParaView-v5.6.1/build
-DTTK_INSTALL_PLUGIN_DIR=/usr/local/lib/plugins
-DPYTHON_NUMPY_INCLUDE_DIR=~/miniconda/envs/p37/lib/python3.7/site-packages/numpy/core/include
-DPYTHON_INCLUDE_DIR=/Users/razoumov/miniconda/envs/p37/include/python3.7m
-DPYTHON_LIBRARY=/Users/razoumov/miniconda/envs/p37/lib/libpython3.7m.dylib
-DTTK_ENABLE_SCIKIT_LEARN=ON
-DQt5_DIR=/usr/local/Cellar/qt/5.15.1/lib/cmake/Qt5
```

Running TTK in Docker: alternative, painless solution

An easy way to use TTK without manual compilation of TTK, ParaView, and their dependencies

1. Install Docker Desktop (Linux, Mac, Windows) and run a pre-built Docker image with pvserver + TTK on local port 11111

   ```
   docker run -it --rm -p 11111:11111 -v "$HOME:/home/`whoami`/" --user $UID \
          topologytoolkit/ttk:5.6.1-stable

   docker run -it --rm -p 11111:11111 -v "$HOME:/home/`whoami`/" --user $UID
          topologytoolkit/ttk:5.8.1-dev
   ```

2. Launch a ParaView client (same major version!) on your laptop and connect to port
   `localhost:11111`
   - all your server-side TTK plugins will be available on the client!

Later in this webinar I will demo running client-server ParaView + TTK from a Singularity container on Cedar

Scalar function in 2D

Toy data similar to `https://topology-tool-kit.github.io/persistentHomologyDummies.html`
On presenter's notebook `~/ttk/mydata/multiGaussianAnalysis.pvsm`

Let's create some toy data: three 2D Gaussians $f(\vec{r}) = e^{-\frac{|\vec{r}-\vec{r}_0|^2}{2\sigma^2}}$, $\quad \vec{r} \in \mathbb{R}^2$

1. Apply Sources | **Plane** at 100^2 resolution, turn on the Axes Grid

2. Apply three successive **Python Calculator**s with three Gaussians

   ```
   p1 = exp(-((inputs[0].Points[:,0]-0.28)**2+(inputs[0].Points[:,1]-0.28)**2)/(2*0.02))
   p2 = 0.8*exp(-((inputs[0].Points[:,0]+0.25)**2+(inputs[0].Points[:,1]+0.3)**2)/(2*0.04))
   p3 = 0.6*exp(-((inputs[0].Points[:,0]+0.35)**2+(inputs[0].Points[:,1]-0.25)**2)/(2*0.03))
   ```

3. Apply non-Python Calculator p=p1+p2+p3

4. Apply **Tetrahedralize** to triangulate the plane (we'll use these triangles to generate the 3D terrain)
   - WarpByScalar is fast but produces inaccurate results for topological analysis

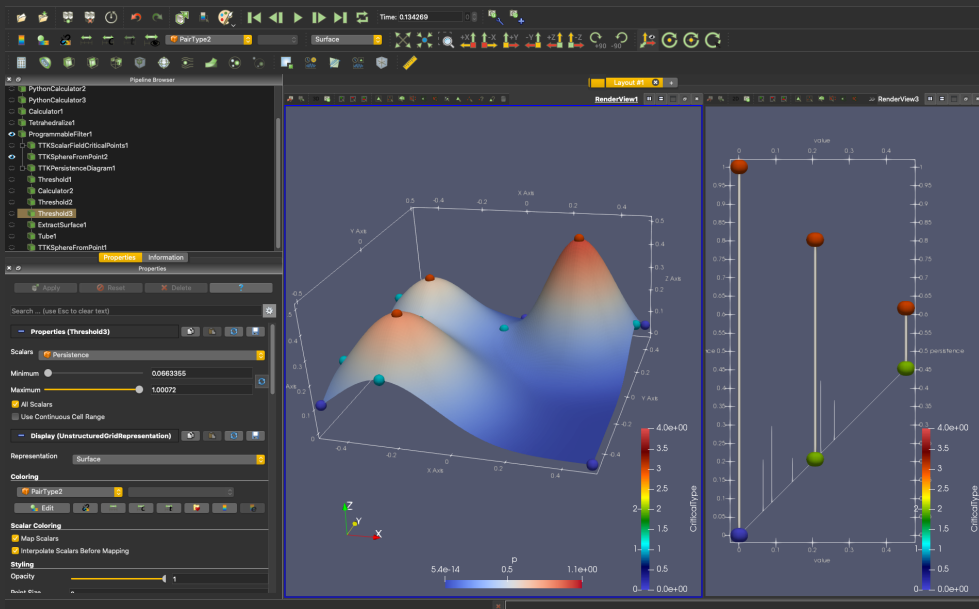5. Apply **ProgrammableFilter** with

   ```
   output.Points[:,2] = 0.5*inputs[0].PointData["p"]      # set the output's z-axis
   output.PointData.append(inputs[0].PointData["p"], "p") # add the data array 'p' to our output
   ```

6. To ProgrammableFilter apply **TTKScalarFieldCriticalPoints**

7. Apply **TTKSphereFromPoint**, set Radius, colour by CriticalType

Intro
○○○○○○○

Installing TTK
○○○

Workflows
○●○○○○○○○○○○○

Cluster demo
○○

Summary
○

Scalar function in 2D (cont.)

8. Create a new render view

9. Apply **PersistenceDiagram**, check Axes Grid, edit X Title = value, Y Title = persistence
   - we'll use its output variables PairIdentifier, PairType, and Persistence

10. Apply **Threshold** on PairIdentifier, keep only non-negative

11. Apply **Calculator** on CellData with PairType2 = abs(PairType)
    - we want to get rid of the "spurious" pairs created by boundary points, i.e. keep PairType=±1
    - PairType=-1 is the largest persistence pair, PairType=0 are spurious/edge pairs (want to exclude these), PairType=1 are the real internal pairs

12. Apply **Threshold** on PairType2, keep only PairType2=1

13. Apply **Threshold** on Persistence, keep all values for now

14. Show the diagonal

15. Apply **ExtractSurface** and then **Tube**

16. To last Threshold apply **TTKSphereFromPoint**, set Radius, colour by CriticalType

Intro
ooooooo

Installing TTK
ooo

Workflows
ooo●ooooooooooo

Cluster demo
oo

Summary
o

Now apply **Threshold** to ProgrammableFilter, vary its *min* value, and check for connected components
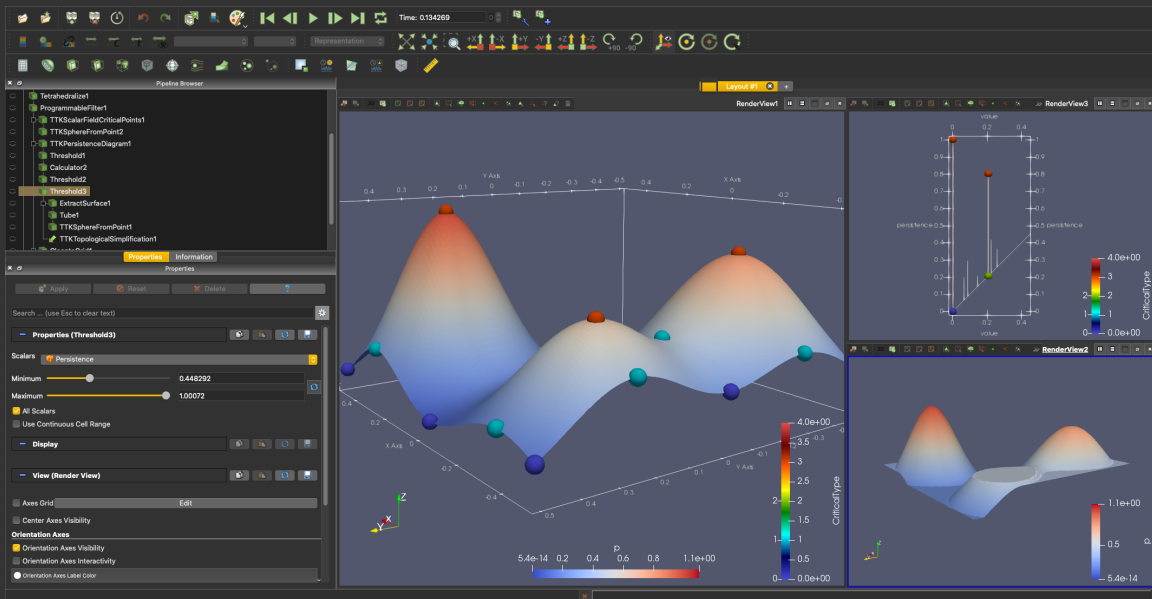
Topological simplification on 2D scalar function

On presenter's notebook ~/ttk/mydata/multiGaussianSimplification.pvsm

1. Start with the last pipeline ~/ttk/mydata/multiGaussianAnalysis.pvsm

2. Open a new render window

3. To ProgrammableFilter apply **CleanToGrid**

4. Apply **TTKTopologicalSimplification** with 2 inputs: Domain = CleanToGrid, Constraints = the last Threshold, scalar field = p, check Numerical Perturbation

5. Apply **ProgrammableFilter** (WarpByScalar does not work here)

```
output.Points[:,2] = 0.5*inputs[0].PointData["p"]
output.PointData.append(inputs[0].PointData["p"], "p")
```

6. Colour by p

7. Modify persistence range (set min=0.45) in the last Threshold

Intro
ooooooo

Installing TTK
ooo

Workflows
ooooo●ooooooooo

Cluster demo
oo

Summary
o

Topological simplification on 2D scalar function (cont.)

Intro
∘∘∘∘∘∘∘

Installing TTK
∘∘∘

Workflows
∘∘∘∘∘●∘∘∘∘∘∘∘

Cluster demo
∘∘

Summary
∘

Extracting the TTK Morse Smale Complex from a dataset

TTKMorseSmaleComplex filter
provides information about critical
points and separatrices that link
these points ⇒ segmentation of
the dataset

1. Load just the 2D function
   `multiGaussianOnly.pvsm`

2. To ProgrammableFilter apply
   TTKMorseSmaleComplex

3. To Critical Points apply
   TTKSphereFromPoint, colour by
   CellDimension

4. To 1-Separatrices apply
   ExtractSurface and then **Tube**

5. Colour Segmentation by
   AscendingManifold or
   DescendingManifold
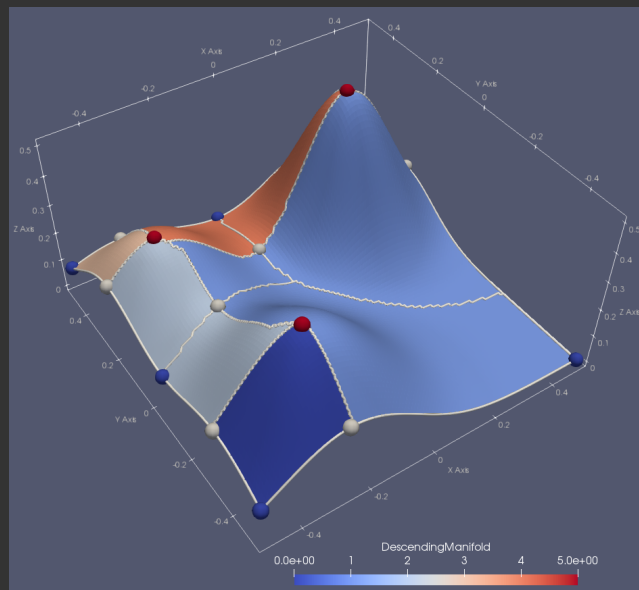
Noisy 2D terrain

- TTK cover visualization: example from TTK tutorials
  `ppp --state=ttk/ttk-data-0.9.8/states/morsePersistence.pvsm`

- Flat plateau on the persistence curve separating noise from the data

- To show the result of topological simplification with the 3D surface, apply **ProgrammableFilter** to the output of TTKTopologicalSimplification making sure to pass `Blend` variable
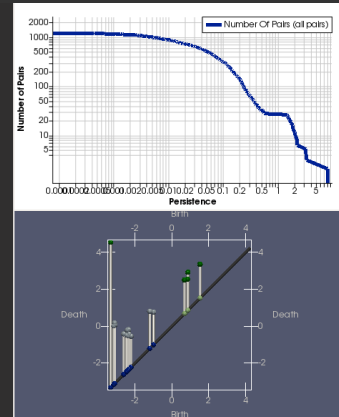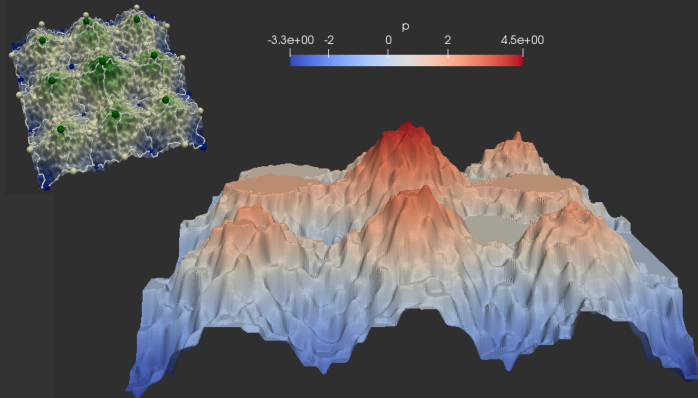
3D scalar field segmentation
This slide is based on Charles Gueunet's "3D foot segmentation" tutorial (Kitware)

1. Load the 3D scan of a human foot `ttk_tutorial_data/foot.vti`
   - load only Scalars field, switch to volumetric rendering

2. Locate 5 largest persistence pairs
   - apply **TTKPersistenceCurve**, close all panes but one
   - show saddle-maximum pairs
   - uncheck Use Index For XAxis, X Array Name = Persistence (x-axis)
   - Series Parameters = Number Of Pairs (y-axis)
   - check Left Axis Log Scale
   - if want to keep 5 pairs ⇒ need to use persistence = 187

3. Use topological simplification on the persistence diagram to extract only 5 maxima
   - open a new render view
   - apply **TTKPersistenceDiagram** to the original data
   - apply **Threshold** on Persistence, click reset, remove all the pairs below 187

4. Remove the diagonal in the persistence diagram
   - apply **Threshold** on PairIdentifier, click reset, remove all negative pairs (min=0)
   - ⇒ 5 vertical lines
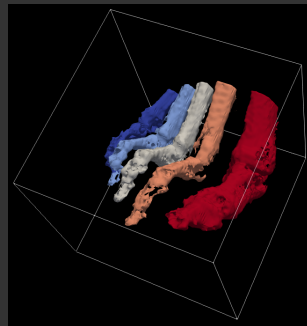
3D scalar field segmentation (cont.)

5. Optionally beautify our persistence diagram
   - add the axes, apply **ExtractSurface** and then **Tube**

6. Perform Topological simplification on the 3D scalar field
   - apply **TTKTopologicalSimplification** with 2 inputs: Domain = foot.vti, Constraints = the first Threshold
   - ⇒ now we have a new version of the foot: no visual difference, but the small noise has been removed
   - ⇒ we want to extract bones, identify toes ⇒ need to show areas attached to maxima ⇒ use a merge tree

7. Compute the split tree of the maxima
   - click on TTKTopologicalSimplification
   - apply **TTKMergeandContourTreeFTM** (FTM = Fibonacci Task-based Merge tree), in Output Options set Tree Type = Split Tree
   - ⇒ we get 5 leaves = toes

8. Attach 3D regions to the leaves
   - click on Segmentation output of the tree in the browser
   - apply **Threshold** on Region Type keeping only the value of 1 (regions attached to the leaves)

9. Beautify the segmented regions
   - apply **ExtractSurface** and then **TTKGeometrySmoother**
   - optionally apply **GenerateSurfaceNormals** to provide smooth shading of the dataset

More complex version of this pipeline:

```
cd ~/ttk/ttk_tutorial_data && ppp --state=tutoSegmentation.pvsm
```

Intro
oooooooo

Installing TTK
ooo

Workflows
oooooooooo●ooo

Cluster demo
oo

Summary
o

3D scalar field segmentation: same algorithm, simpler dataset

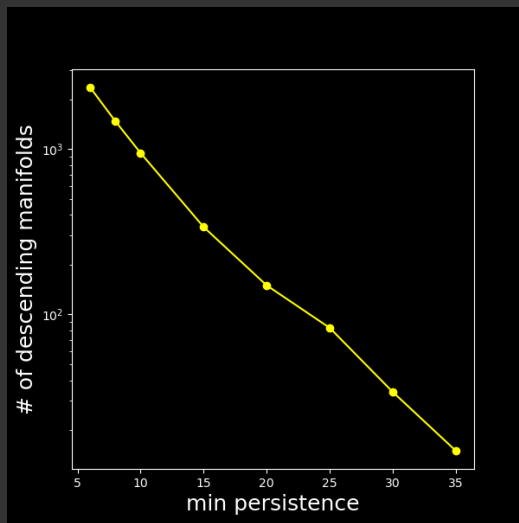On presenter's notebook `~/ttk/mydata/stvol.pvsm`

Discretized 3D Styblinski-Tang function inside a unit cube ($x_i \in [0, 1]$)

$$f(x_1, x_2, x_3) = \frac{1}{2} \sum_{i=1}^{3} (\xi_i^4 - 16\xi_i^2 + 5\xi_i), \text{ where } \xi_i \equiv 8(x_i - 0.5)$$
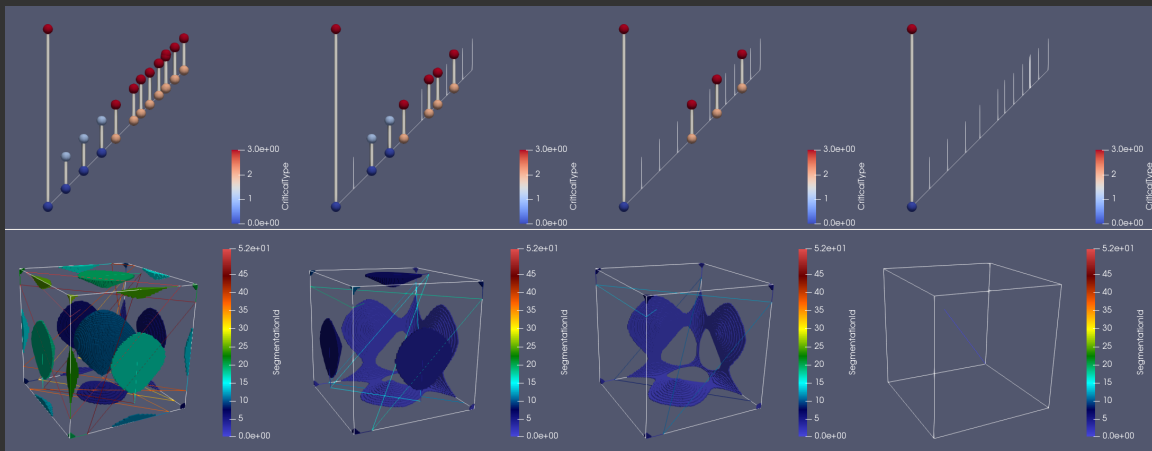
Image segmentation using topological persistence

- Area of active research

- Use TTK as a computational tool to test your ideas

- TTK sample datasets include one example

- I modified the workflow: simplied, added a persistence curve – let's take a look at the algorithm

```
cd ~/ttk/mydata
ppp --state=imageProcessingModified.pvsm
```

Intro
0000000

Installing TTK
000

Workflows
0000000000000●0

Cluster demo
00

Summary
0

Image segmentation using topological persistence (cont.)

Let's apply the same technique to another image

1. Copy `imageProcessingModified.pvsm` into a new state file

2. Edit it, replacing (2X) `naturalImage.png` with your new image's name
   - here we'll use a scaled-down aurora image from
     `https://en.wikipedia.org/wiki/Image_segmentation`

3. Load your new state file into ParaView

4. Modify (1) the persistence min/max thresholds and (2) the colour range in the segmented image
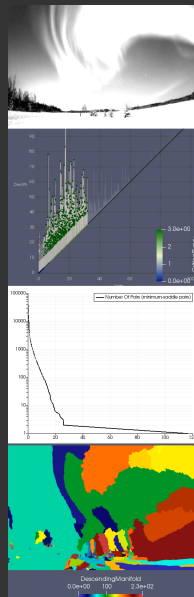
Image segmentation using topological persistence (cont.)

- In the previous two examples the key was to use the PNG image's (which filter?) gradient as the scalar function

- This *won't work* for all images, so it is your task as a researcher to find a better scalar function, to be used for topological image segmentation

- Active research area

Image segmentation using topological persistence (cont.)

- In the previous two examples the key was to use the PNG image's (which filter?) gradient as the scalar function

- This *won't work* for all images, so it is your task as a researcher to find a better scalar function, to be used for topological image segmentation

- Active research area

- More complex example from TTK tutorials: cell enumeration in confocal microscopy, based on the example from Herbert Edelsbrunner's and John Harer's book "Computational Topology: An Introduction", AMS (2009)

  ```
  cd ~/ttk/ttk-data-0.9.8 && ppp --state=states/tribute.pvsm
  ```

  - very noisy input image
  - different scalar function

Intro
ooooooo

Installing TTK
ooo

Workflows
ooooooooooooo

Cluster demo
●o

Summary
o

Running client-server ParaView + TTK via Singularity on Cedar

1. Build ParaView 5.6.2 + TTK Docker in Ubuntu 20.04 image on Cedar
   `https://docs.computecanada.ca/wiki/Singularity`

```
$ cd ~/scratch
$ wget https://raw.githubusercontent.com/moby/moby/\
    master/contrib/download-frozen-image-v2.sh
$ sh download-frozen-image-v2.sh build \
    topologytoolkit/ttk:5.6.2-stable   # download an image from Docker Hub into build/
$ cd build && tar cvf ../topologytoolkit.tar * && cd ..
$ salloc --mem-per-cpu=3600 --cpus-per-task=2 --time=0:15:0 --account=cc-debug
$ module load singularity
$ singularity build topologytoolkit562.sif \
    docker-archive://topologytoolkit.tar   # build the Singularity image
$ /bin/rm -rf build topologytoolkit.tar
```

   - TTK source code includes scripts for building Docker and Singularity images from scratch
   - alternatively can use topologytoolkit/ttk:5.8.1-dev to build topologytoolkit581.sif
     (it did not work for me ...)

Running client-server ParaView + TTK via Singularity on Cedar (cont.)

2. Start ParaView server on a compute node

```
$ cd ~/scratch
$ salloc --mem-per-cpu=3600 --time=0:30:0 --account=cc-debug
$ module load singularity
$ singularity run -B /home -B /scratch topologytoolkit562.sif pvserver
```

3. Connect from the ParaView client on your computer

```
$ ssh cedar.computecanada.ca -L 11111:cdr768:11111
start ParaView 5.6.x on your computer and connect to localhost:11111
File | Load State - local ~/ttk/ttk-data-0.9.8/states/dragon.pvsm
Choose File Names - remote cedar:/scratch/razoumov/dragon.vtu
```

Intro
○○○○○○○

Installing TTK
○○○

Workflows
○○○○○○○○○○○○

Cluster demo
○○

Summary
●

Questions?