TopoVol

Topology driven volume rendering with TTK

Qi Wu (Prof. Kwan-Liu Ma, University of California - Davis)

Email: qadwu@ucdavis.edu

VIS 2018

TopoVol: What and Why

Transfer Function Classification

+

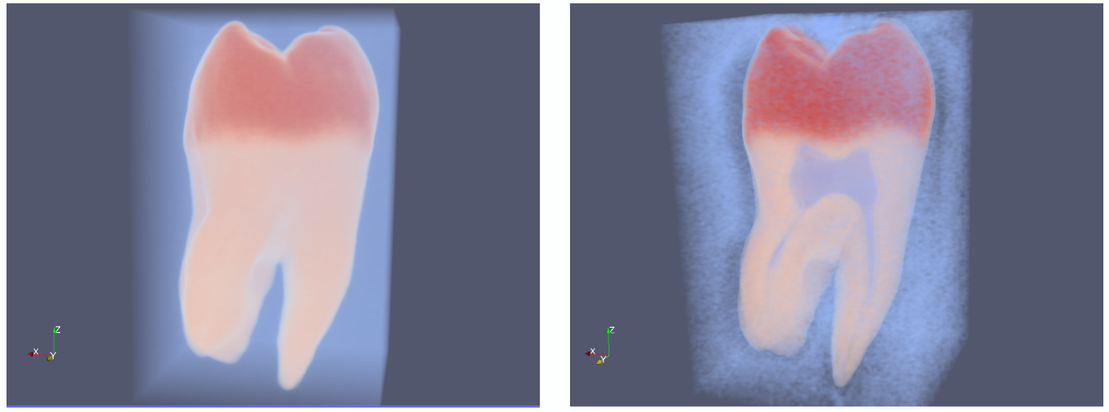
Topological Segementation

=

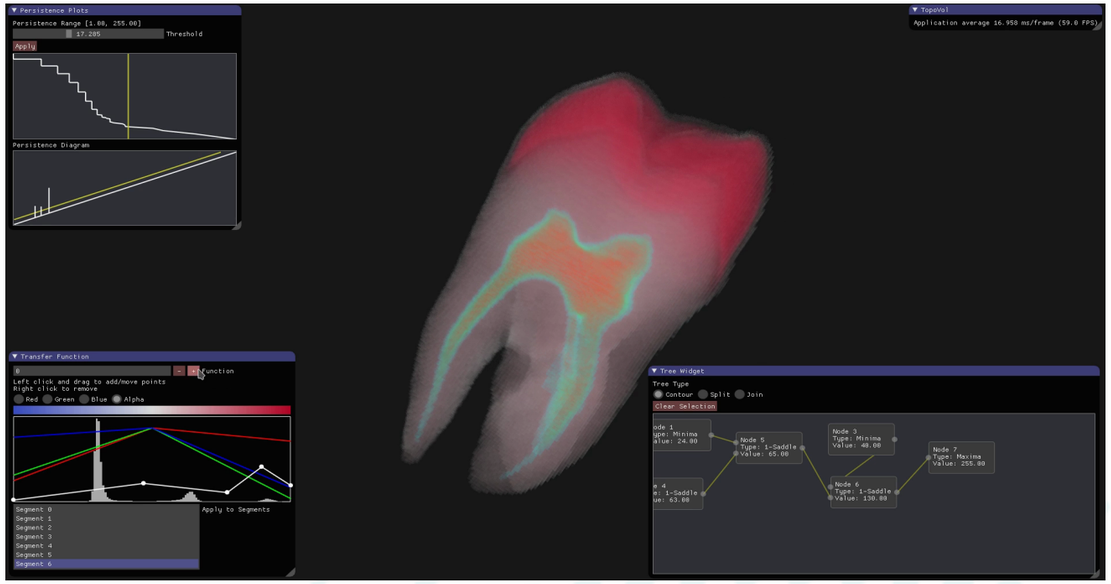
TopoVol

OpenGL + VTK + TTK

An example showing how to bring TTK into your apps



Single Transfer Function

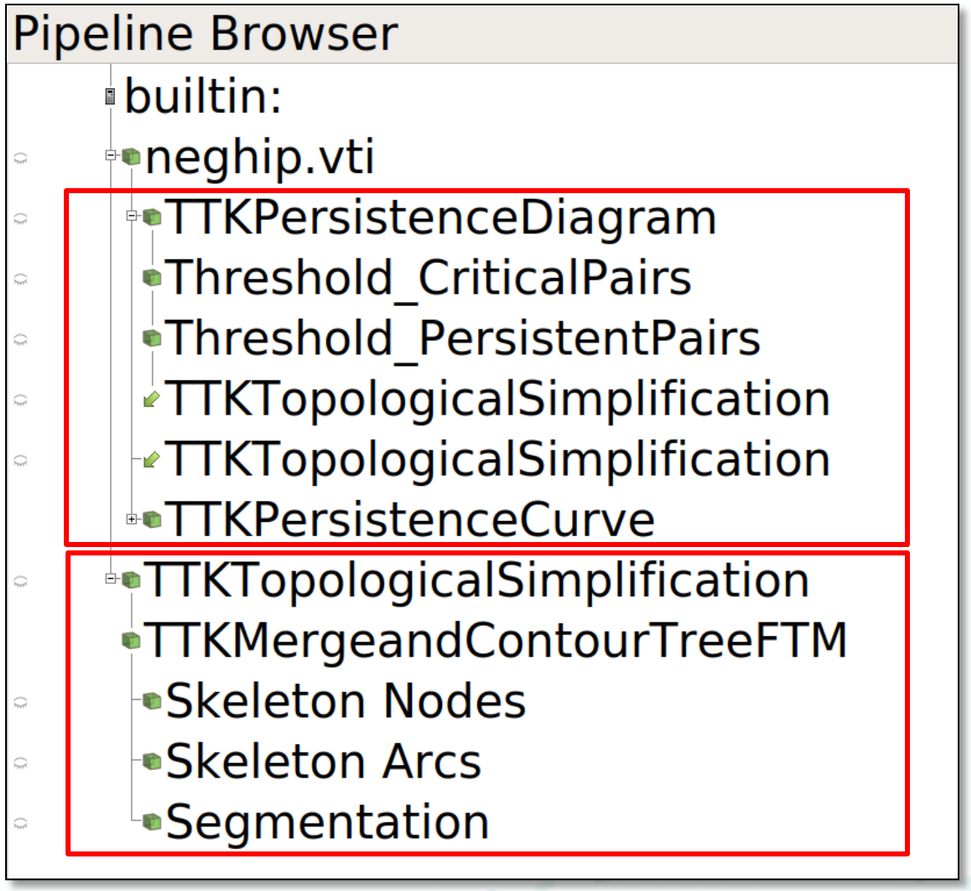


TopoVol

VIS 2018

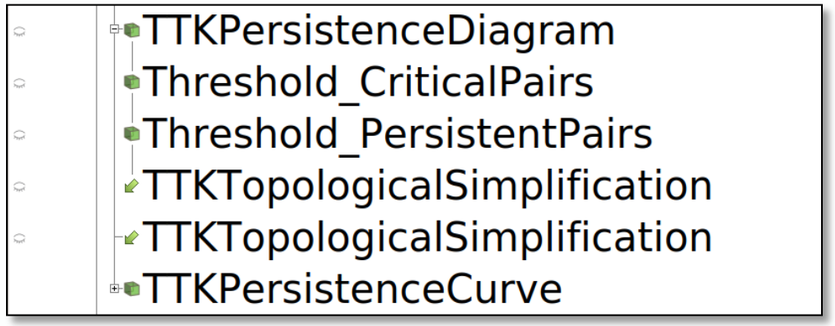
Construct a VTK/TTK Pipeline

- Construct a working pipeline ParaView
  - Persistence curve & diagram
  - Topological Simplification
  - Contour/Split/Merge Tree
- Translate the pipeline into code with the help of ParaView

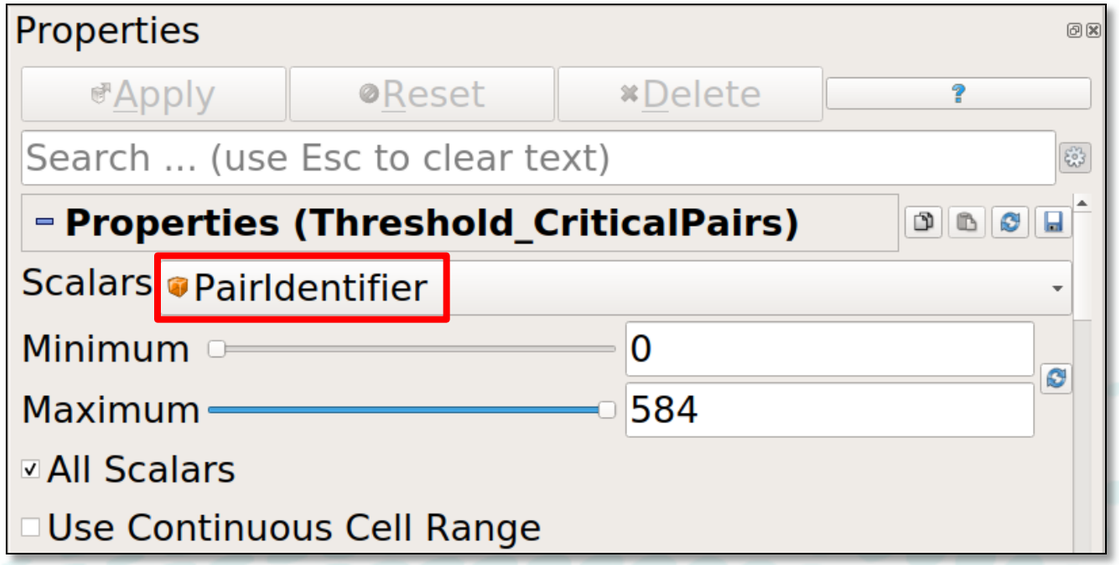


Computing the Persistence Diagram & Curve

```
// Compute TTK persistence diagram
auto diagram = vtkSmartPointer<ttkPersistenceDiagram>::New();
configure_ttk(diagram.Get());
diagram->SetInputData(data);

// Compute critical points
auto critical_pairs = vtkSmartPointer<vtkThreshold>::New();
critical_pairs->SetInputConnection(diagram->GetOutputPort());
critical_pairs->SetInputArrayToProcess(0, 0, 0,
    vtkDataObject::FIELD_ASSOCIATION_CELLS, "PairIdentifier");
critical_pairs->ThresholdBetween(-0.1, max_value);
```



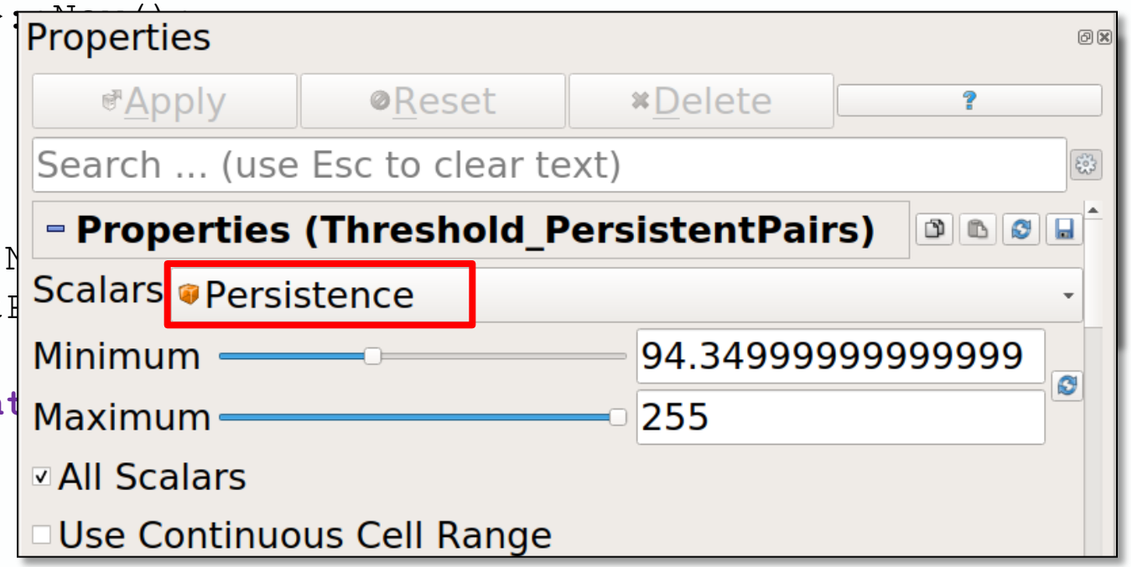

```
void configure_ttk(ttk::Debug *obj) {
    obj->SetUseAllCores(true);
    obj->SetThreadNumber(numThreads);
    obj->SetdebugLevel_(debuglevel);
}
```



VIS 2018

Computing the Persistence Diagram & Curve

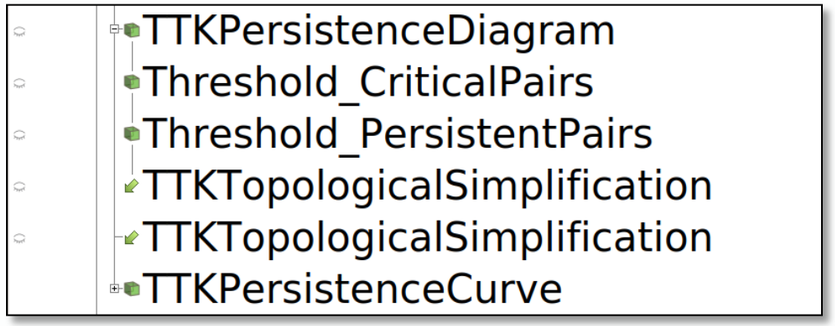
```
// Compute TTK persistence diagram
auto diagram = vtkSmartPointer<ttkPersistenceDiagram>::New();
configure_ttk(diagram.Get());
diagram->SetInputData(data);

// Compute critical points
auto critical_pairs = vtkSmartPointer<vtkThreshold>::N
critical_pairs->SetInputConnection(diagram->GetOutputP
critical_pairs->SetInputArrayToProcess(0, 0, 0,
    vtkDataObject::FIELD_ASSOCIATION_CELLS, "PairIdent
critical_pairs->ThresholdBetween(-0.1, max_value);

// Select the most persistent pairs
auto persistent_pairs = vtkSmartPointer<vtkThreshold>::New();
persistent_pairs->SetInputConnection(critical_pairs->GetOutputPort());
persistent_pairs->SetInputArrayToProcess(0, 0, 0,
    vtkDataObject::FIELD_ASSOCIATION_CELLS, "Persistence");
persistent_pairs->ThresholdBetween(threshold,999999);
```

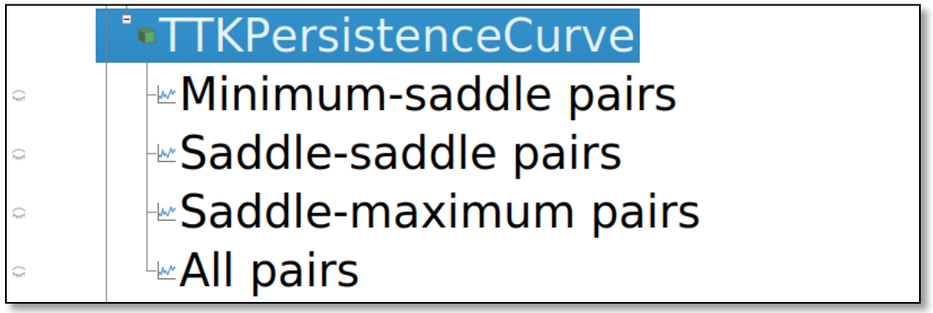


VIS 2018

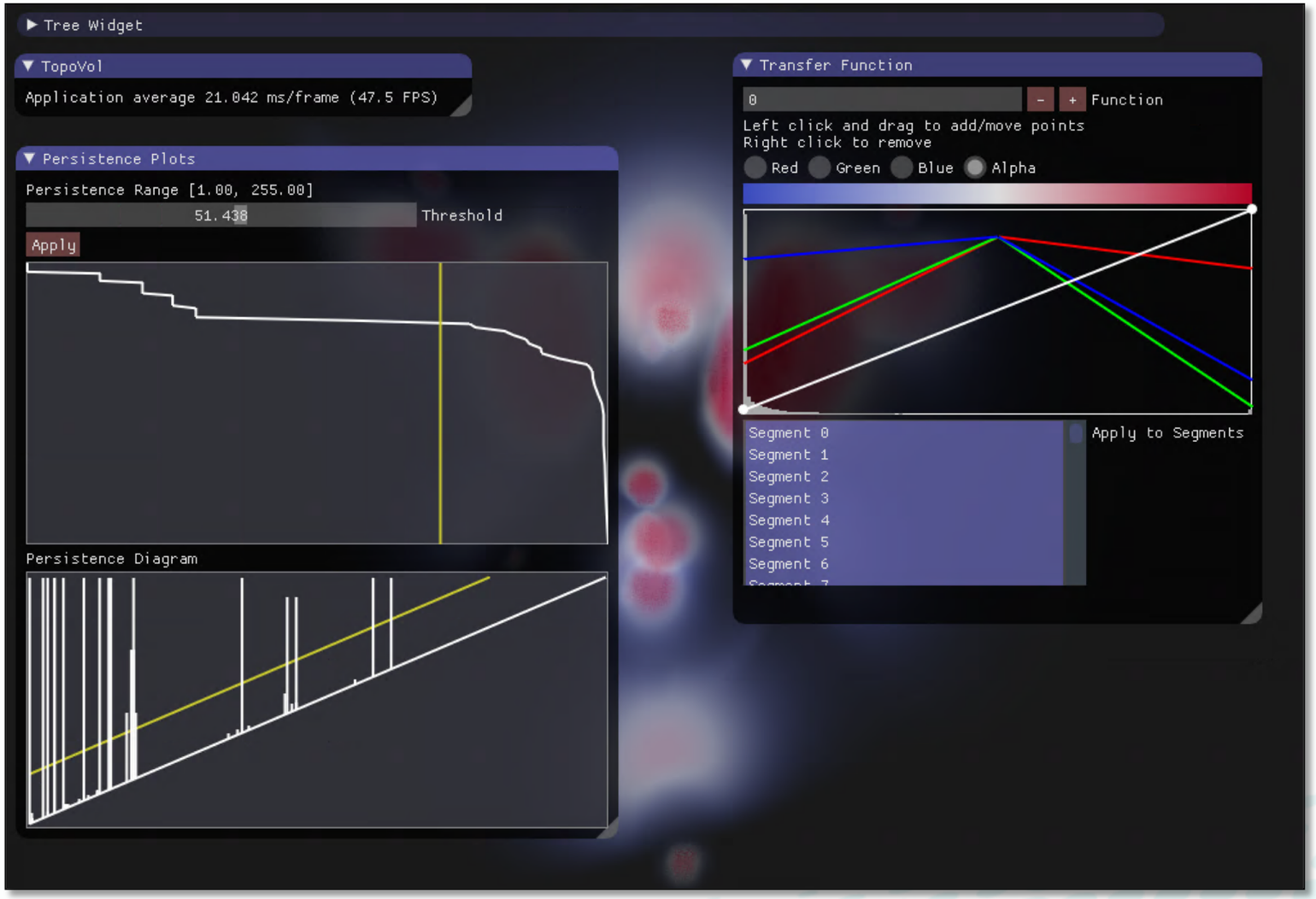
Computing the Persistence Diagram & Curve

```
// We always show the full curve, without simplification for the
// selected tree type
auto vtkcurve = vtkSmartPointer<ttkPersistenceCurve>::New();
configure_ttk(vtkcurve.Get());
vtkcurve->SetInputData(data);
vtkcurve->SetComputeSaddleConnectors(false);

//...

// Update the VTK pipeline
vtkcurve->Update();
```





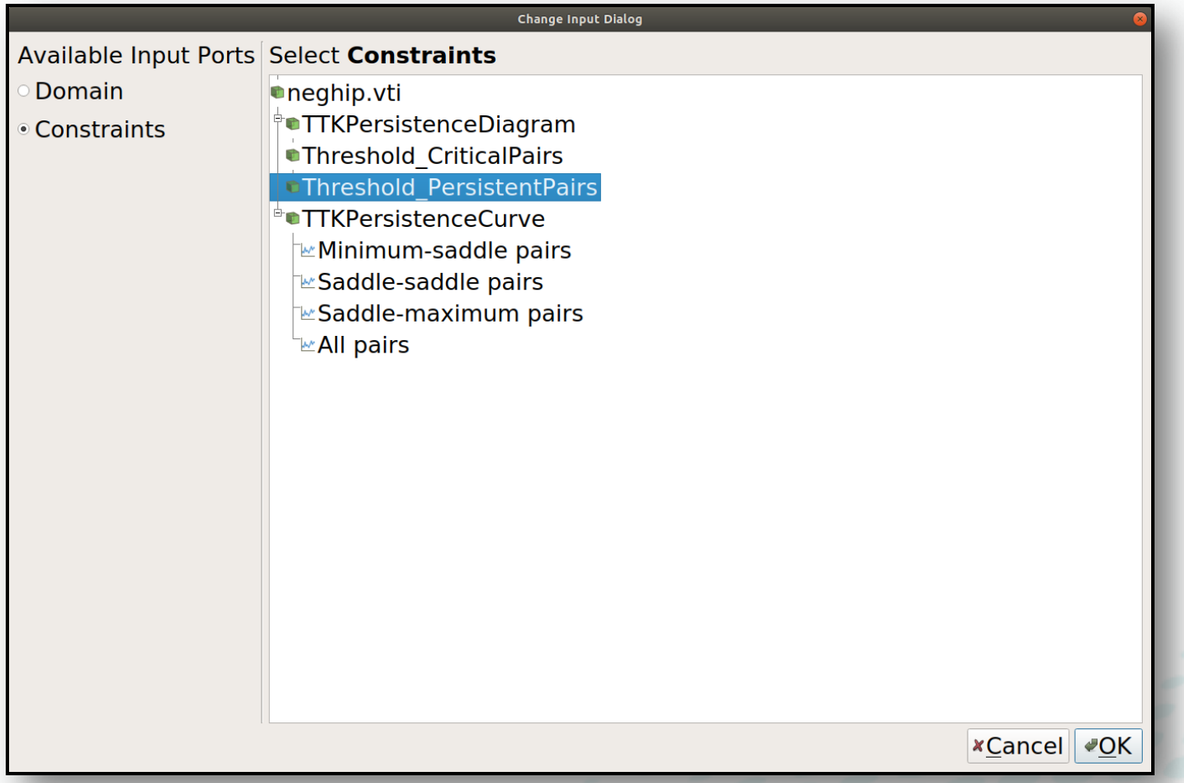
Computing the Persistence Diagram & Curve



VIS 2018

Applying the Persistence Simplification

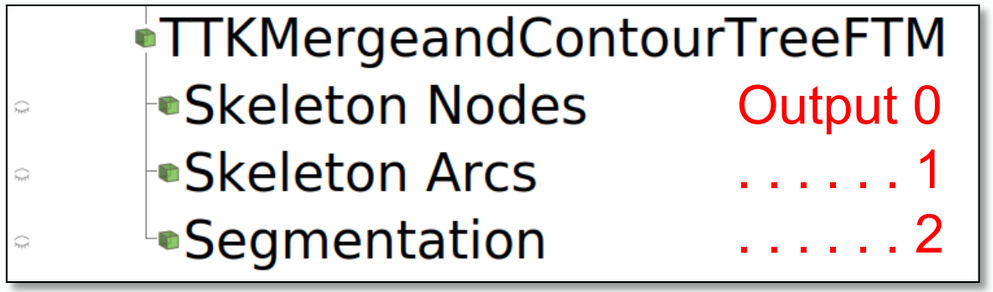
```
// Smooth the volume to reduce the number of pairs
auto simplification = vtkSmartPointer<ttkTopologicalSimplification>::New();
configure_ttk(simplification.Get());
simplification->SetInputData(0, data);
simplification->SetInputConnection(1, persistent_pairs->GetOutputPort());
```

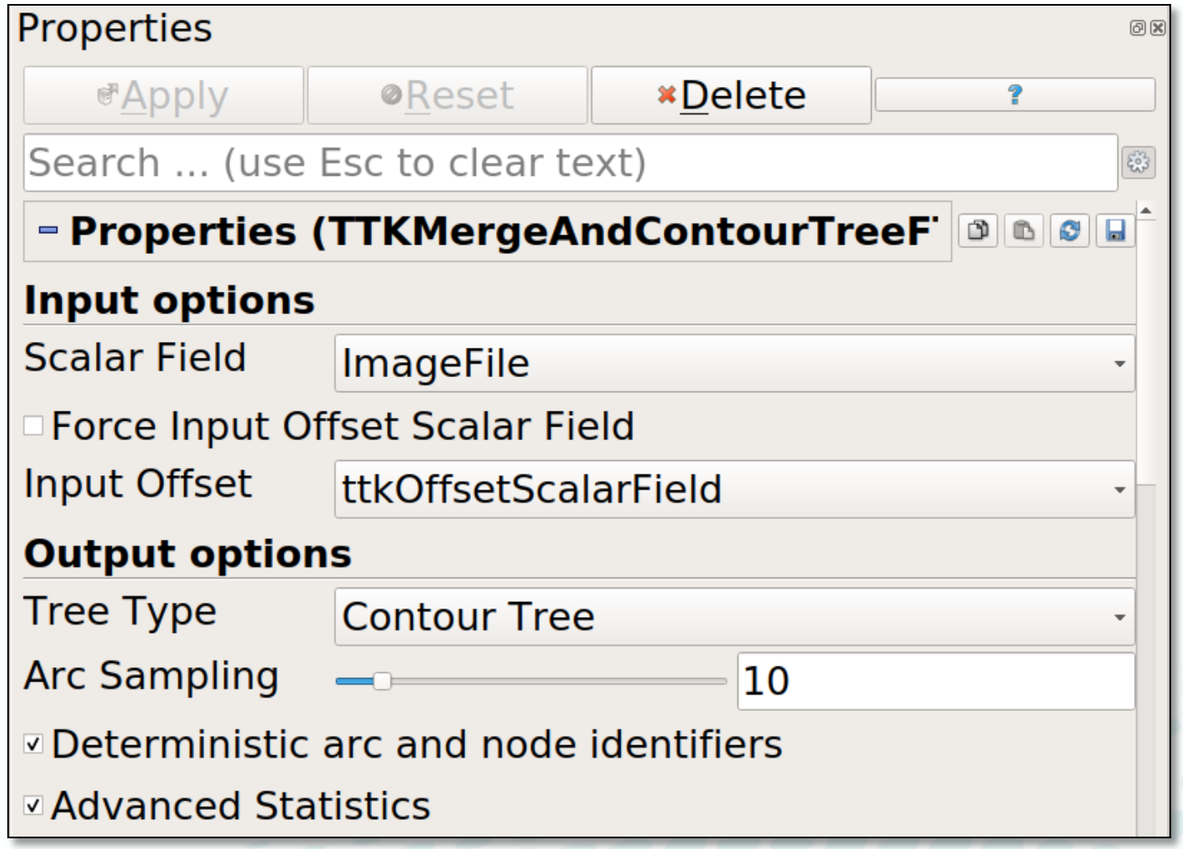


Fibonacci Task-based Merge Tree [1]

Computing FTM Tree

```
auto tree = vtkSmartPointer<ttkFTMTree>::New();
configure_ttk(tree.Get());
tree->SetInputConnection(simplification->GetOutputPort());
tree->SetSuperArcSamplingLevel(10);
tree->SetWithSegmentation(true);
```

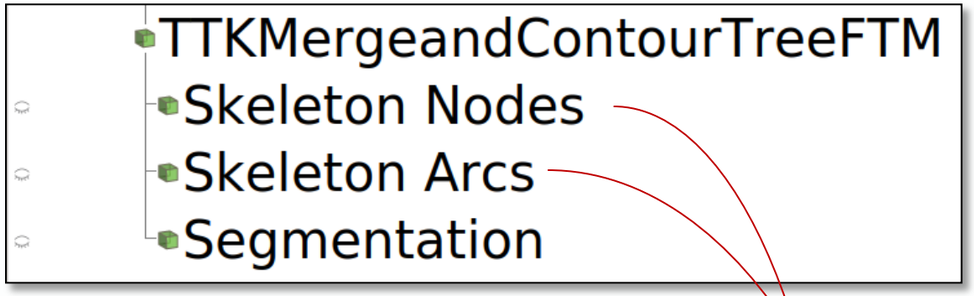




VIS 2018

[1] Gueunet et. al. LDAV 2017

Computing FTM Tree



Information

Statistics

Type: Unstructured Grid
Number of Cells: 0
Number of Points: 38
Memory: 0.024 MB

Data Arrays

Name	Data Type	Data Ranges
NodeId	int	[0, 37]
NodeType	int	[0, 4]
RegionSize	int	[23, 175982]
RegionSpan	int	[1, 53]
Scalar	float	[0, 255]
VertexId	int	[0, 236962]

Bounds

X Range: 0 to 59 (delta: 59)
Y Range: 0 to 54 (delta: 54)
Z Range: 0 to 57 (delta: 57)

Time

Information

Statistics

Type: Unstructured Grid
Number of Cells: 337
Number of Points: 338
Memory: 0.046 MB

Data Arrays

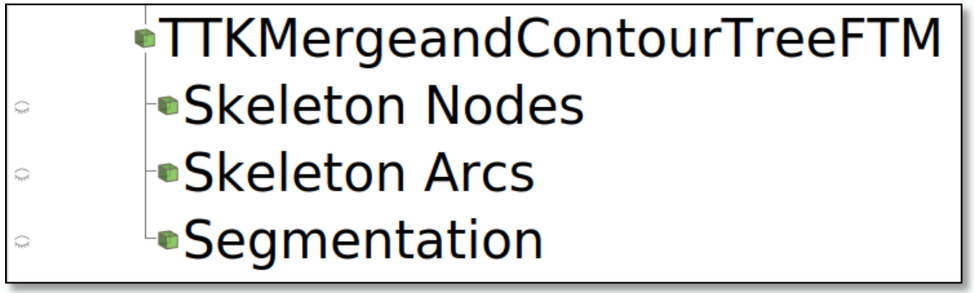
Name	Data Type	Data Ranges
RegularMask	char	[0, 1]
Scalar	float	[0, 255]
downNodeId	int	[1, 37]
RegionSize	int	[23, 175982]
RegionSpan	double	[1.73205, 53.1413]
SegmentationId	int	[0, 36]
upNodeId	int	[0, 36]

Bounds

X Range: 0 to 59 (delta: 59)
Y Range: 0 to 54 (delta: 54)
Z Range: 0 to 57.5 (delta: 57.5)

```
tree_nodes = dynamic_cast<vtkUnstructuredGrid*>(tree->GetOutput(0));
tree_arcs  = dynamic_cast<vtkUnstructuredGrid*>(tree->GetOutput(1));
```

Computing FTM Tree



Showing TTKMergeandContourTreeFTM (Skeleton Nodes) Attribute: Point Data Precision: 6

	Point ID	NodeId	NodeType	Points			RegionSize	RegionSpan	Scalar	VertexId
0	0	0	4	34	54	57	23	1	255	236962
1	1	1	2	35	53	56	2676	17	24	232803
2	2	2								
3	3	3								
4	4	4								
5	5	5								
6	6	6								
7	7	7								
8	8	8								
9	9	9								
10	10	10								

Showing TTKMergeandContourTreeFTM (Skeleton Arcs) Attribute: Point Data Precision: 6

	Point ID	Points			RegularMask	Scalar
0	0	35	53	56	0	24
1	1	34	54	57	0	255
2	2	33.2				
3	3	32.5				
4	4	34.5				
5	5	33.5				
6	6	34				
7	7	32				
8	8	32.4				
9	9	30.5				
10	10	32.6				

Showing TTKMergeandContourTreeFTM (Skeleton Arcs) Attribute: Cell Data Precision: 6

	Cell ID	Cell Type	Point Index 0	Point Index 1	RegionSize	RegionSpan	SegmentationId	downNodeId	upNodeId
0	0	Line	0	2	23	1.73205	36	1	0
1	1	Line	2	3	23	1.73205	36	1	0
2	2	Line	3	4	23	1.73205	36	1	0
3	3	Line	4	5	23	1.73205	36	1	0
4	4	Line	5	1	23	1.73205	36	1	0
5	5	Line	6	8	34	2.44949	35	3	2
6	6	Line	8	9	34	2.44949	35	3	2
7	7	Line	9	10	34	2.44949	35	3	2
8	8	Line	10	11	34	2.44949	35	3	2
9	9	Line	11	12	34	2.44949	35	3	2
10	10	Line	12	13	34	2.44949	35	3	2

Get spreadsheets

```
vtkDataSetAttributes *arc_point_attribs = tree_arcs->GetAttributes(vtkDataSet::POINT);
vtkDataSetAttributes *arc_cell_attribs  = tree_arcs->GetAttributes(vtkDataSet::CELL);
vtkPoints     *arc_points = tree_arcs->GetPoints();
vtkCellArray  *arc_cells  = tree_arcs->GetCells();
```

Get primitives

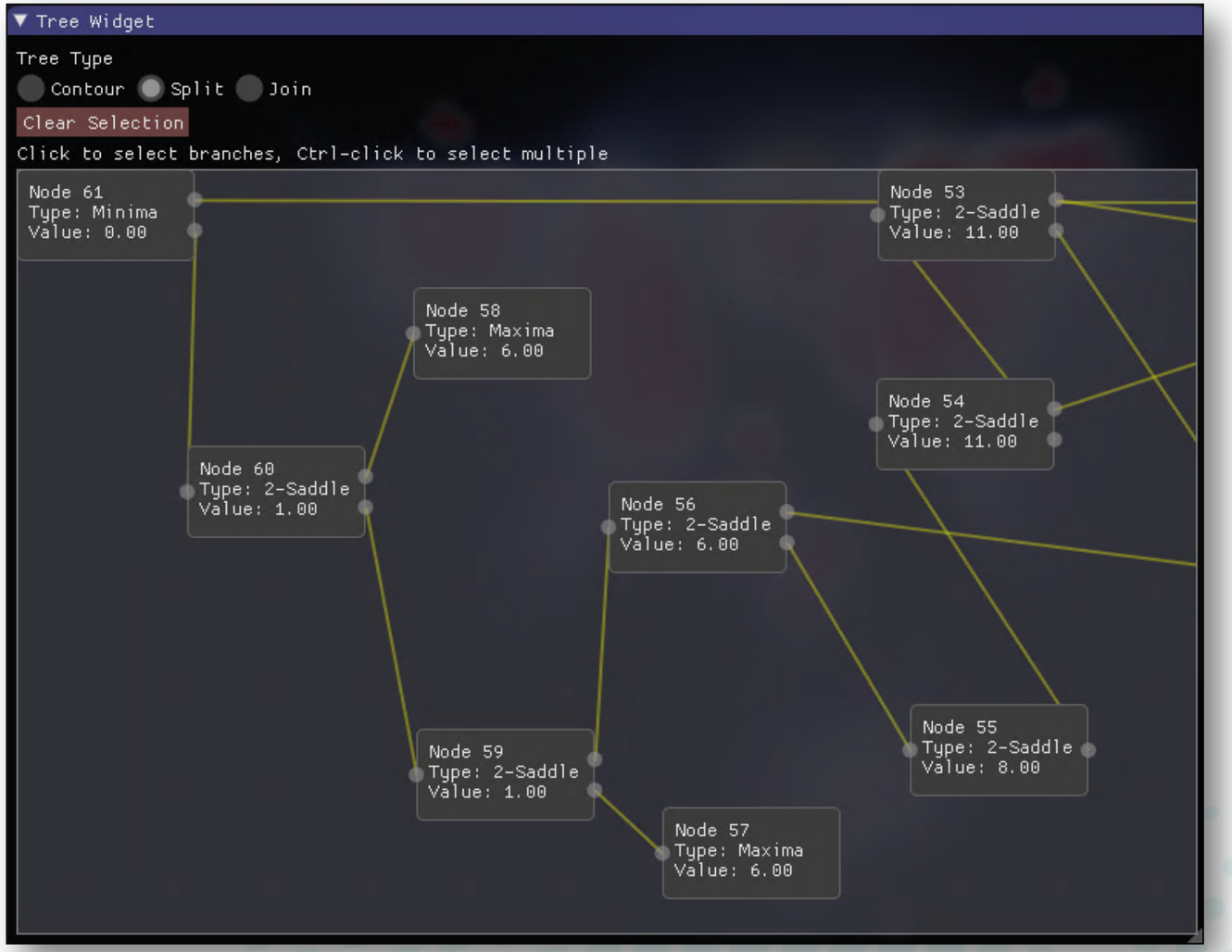
```
auto *line_segmentation_id =
    vtkIntArray::SafeDownCast(arc_cell_attribs->GetArray("SegmentationId"));
```

Computing FTM Tree

```
// Traverse arcs
auto idList = vtkSmartPointer<vtkIdList>::New();
vtkIdType line_idx = 0;
lines->InitTraversal();
while (lines->GetNextCell(idList)) {
    double pt_pos[3];
    points->GetPoint(idList->GetId(0), pt_pos);
    //...
    points->GetPoint(idList->GetId(0), pt_pos);
    //...
    const int32_t seg = line_segmentation_id->GetValue(line_idx);
    const float start_val = point_scalar->GetValue(idList->GetId(0));
    const float end_val   = point_scalar->GetValue(idList->GetId(1));
    //...
    ++line_idx;
}
```

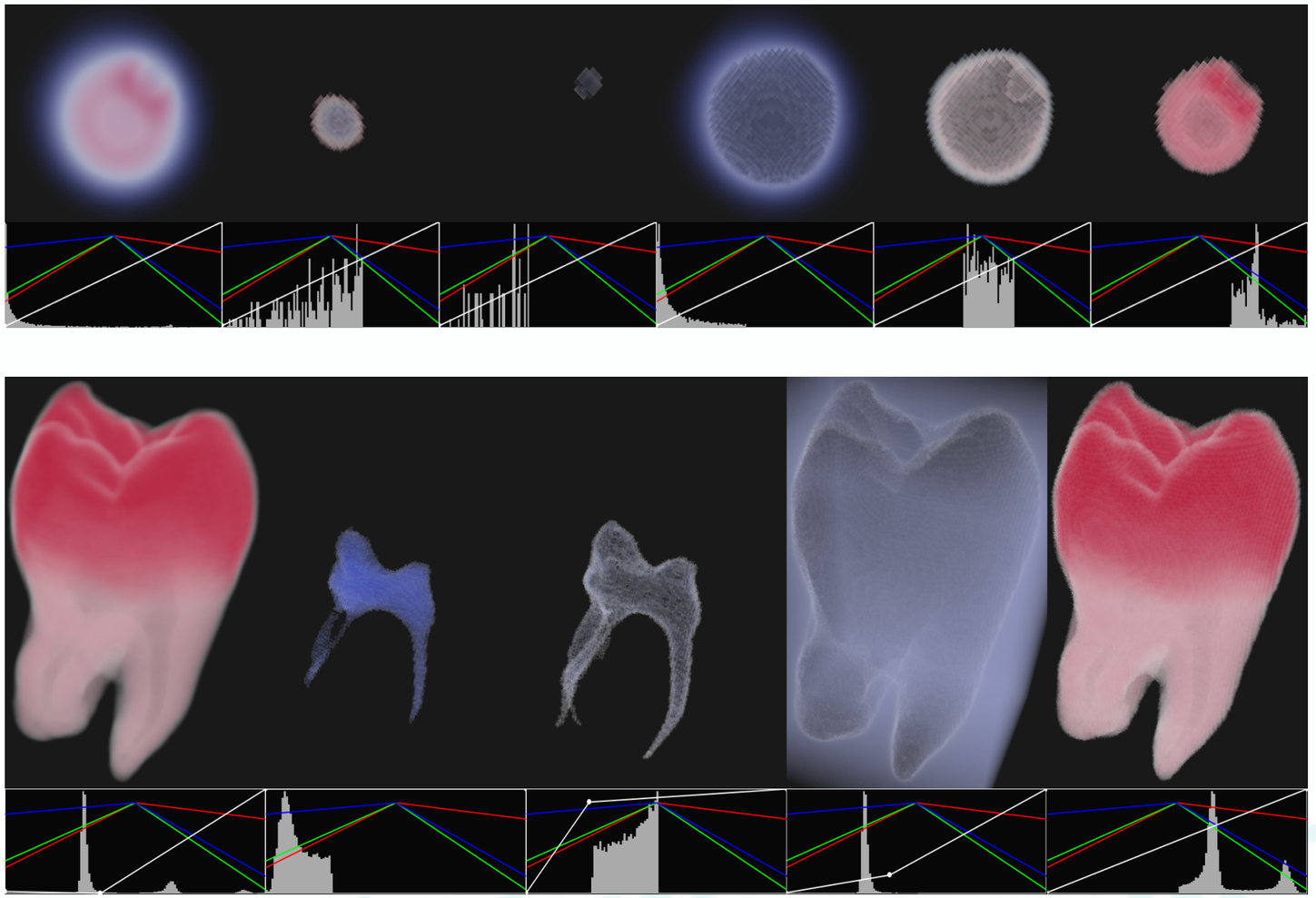
Drawing the FTM Tree UI with Dear ImGui

Construct the tree graph by connecting the nodes and edges using our favorite GUI toolkit: ImGui



Rendering with Multiple Transfer Functions

- Data used in shaders
  - Scalar Volume
  - Segmentation volume
  - A transfer function list
- Steps
  - Check if voxel is selected
  - Find a transfer function for it
  - Color the voxel



Conclusion

- VTK/TTK API is simple to use and capable of being integrated in your application even without being a TTK expert
- Steps to build a VTK/TTK application
  - Construct the pipeline w/ ParaView
  - Implement the ParaView pipeline using VTK/TTK API
  - Remember to use ParaView's GUI interface for debugging (input/output data type, unsure attribute names, etc)

TopoVol git repository https://git.io/fxfxA