



HAL
open science

Simple visualizations of unstructured grids with VTK

Roman Putanowicz, Frédéric Magoulès

► **To cite this version:**

Roman Putanowicz, Frédéric Magoulès. Simple visualizations of unstructured grids with VTK. [Intern report] A03-R-039 || putanowicz03b, 2003, 27 p. inria-00107738

HAL Id: inria-00107738

<https://inria.hal.science/inria-00107738v1>

Submitted on 19 Oct 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Simple visualizations of unstructured grids with VTK

Roman Putanowicz
<putanowr@twins.pk.edu.pl>
Frédéric Magoulès
<frederic.magoules@iecn.u-nancy.fr>

Revision : 1.2
February 26, 2003

Contents

1	Introduction	2
2	Simple data file	2
3	Scalar data visualization	4
4	Complex data file	6
5	Scalar data visualization – cont.	7
6	Scalar data visualization with color bar	9
7	Mesh visualization	11
8	Subdomains visualization	14
9	Labeling points	17
10	Labeling cells	21
11	Saving visualization to a file	26

1 Introduction

This report shows examples of simple visualizations of unstructured grids with use of VTK - The Visualization Toolkit. VTK is a software system for computer graphics, visualization and image processing. Although VTK is a large and complex library, it is possible to write with it short and easily understandable applications, which perform quite complex tasks. This is due to the object-oriented, high level interface provided by VTK, as well as to the possibility of using VTK with languages like Tcl, Java and Python (VTK itself is written in C++).

VTK library is distributed as open-source software with extensive class documentation and lots of examples. The examples are however organized around particular classes, like `vtkPolyDataMapper`, what might be a bit confusing for a beginner, who usually wants to answer simple questions, like: "What should I do to draw a mesh?". This report provides answers to few such simple questions but it is by no means a substitution for reading documentation or literature. For this reasons we do not provide explanation or discussion of several basic concepts like visualization, visualization pipeline, data sets but instead we point to consult specific references [1, 2]. For readers who need introduction to Tcl programming language we suggest books [3, 4].

All presented examples are written using Tcl interface to VTK library. With each example there is a short description, a diagram showing the organization of a visualization pipeline, a Tcl script¹ and results of running it on sample data. The scripts, commented and fairly simple, can be a good starting point for readers to do their own experiments in visualization with VTK.

2 Simple data file

The data file for the first example is called "`stripSimple.vtk`". This file defines geometry and topology of an unstructured grid together with a scalar field called "`temperature`". Numbering of nodes, elements and node coordinates are shown in figure 1. Note the use of VTK convention, in which nodes and elements are numbered starting from 0. The scalar values associated with nodes is shown in figure 2. For the

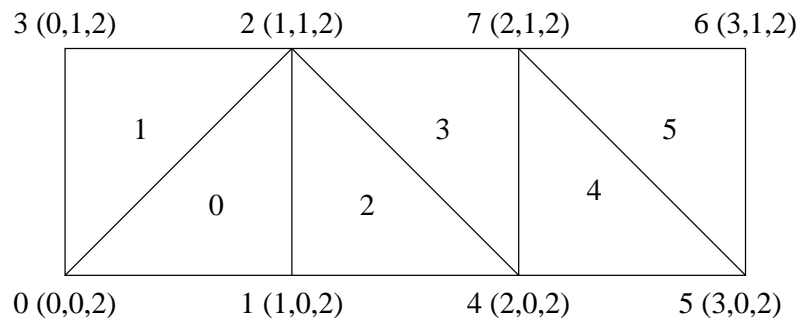


Figure 1: Nodes and elements numbering. Node coordinates are shown in brackets.

data file, we use VTK own data format. The file format consists of five parts :

¹ The complete scripts are shown with line numbers. The numbers are not part of the script.

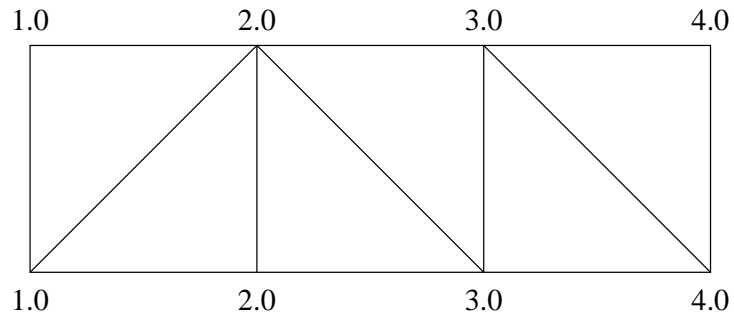


Figure 2: Point attributes "temperature".

1. file version identifier
2. one line header describing the file
3. type of the file (ASCII/BINARY)
4. data set description - geometry and topology
5. data set attributes - can be specified separately for points and for cells

Further details on VTK file format can be found in [1,2].

The data attribute section in "stripSimple.vtk" contains specification of point data as an array of scalar values of type float, called "temperature". That array have one component (out of four allowed). The array is associated with default lookup table (i.e. a table which holds array of colors - for instance RGB (Red-Green-Blue) color components).

The "stripSimple.vtk" file is given below:

```
# vtk DataFile Version 3.0
2D scalar data
ASCII

DATASET UNSTRUCTURED_GRID
POINTS 8 float
0 0 2
1 0 2
1 1 2
0 1 2
2 0 2
3 0 2
3 1 2
2 1 2

CELLS 6 24
3 0 1 2
3 0 2 3
3 1 4 2
```

```

3 4 7 2
3 4 5 7
3 5 6 7

CELL_TYPES 6
5
5
5
5
5
5

POINT_DATA 8
SCALARS temperature float 1
LOOKUP_TABLE default
1.0 2.0 2.0 1.0
3.0 4.0 4.0 3.0

```

3 Scalar data visualization

This example shows most basic visualization technique which is color mapping or in other words assigning colors to elements, based on the scalar value at the points. This technique can be effectively used to visualize scalar fields. The visualization pipeline is shown in the figure 3. In the example, the color mapping is performed by an instance of `vtkDataSetMapper`. `vtkDataSetMapper` receives its input as unstructured grid from `vtkUnstructuredGridReader`.

One of the methods to control color mapping provided by `vtkDataSetMapper` is `SetScalarRange`. That method sets the minimum and maximum value of scalars, that are mapped to colors. Scalars values less than the minimum are clamped to minimum color and scalar values greater than the maximum are clamped to the maximum color. In the line:

```
eval dataMapper SetScalarRange [[reader GetOutput] GetScalarRange]
```

the scalar range is set to be the range of data values in the grid. Note that this line can be rewritten as follows:

```
set drange [[reader GetOutput] GetScalarRange]
set dmin [lindex $drange 0]
set dmax [lindex $drange 1]
dataMapper SetScalarRange $dmin $dmax
```

and if we would like to manually set range (let say from 1.2 to 2.5), we can use the following command:

```
dataMapper SetScalarRange 1.2 2.5
```

The `GetScalarRange` method shown in the above code snippets should be used with care. This method returns the range of scalar value attributes in the grid, but it

considers scalar attributes both in points and in cells. If we want to pass to the mapper the range of scalar field for points only, we need to select point data first and then inquire the scalar range. How it can be done is shown in section 5.

The `vtkDataSetMapper` allows to control many more aspects of the mapping (e.g. chose the field to be mapped, chose points or cell data, etc), but for our simple example the default values and behaviour is enough.

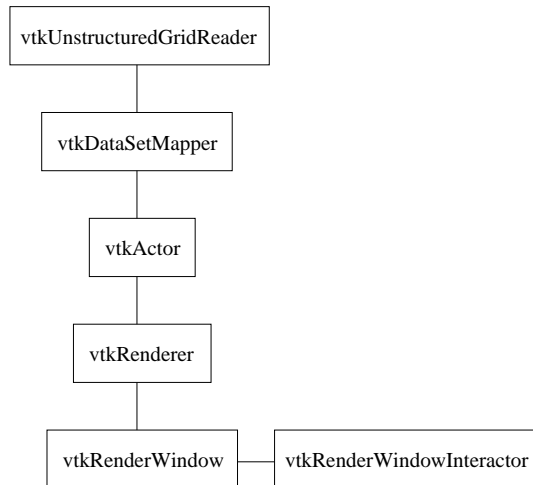


Figure 3: Scalar data visualization pipeline.

```

1  package require vtk
2  package require vtkinteraction

3  # set the name of data file
4  set file "stripSimple.vtk"

5  vtkUnstructuredGridReader reader
6  reader SetFileName "$file"
7  # reader Update is needed because of .. GetScalarRange ..
8  reader Update

9  vtkDataSetMapper dataMapper
10 dataMapper SetInput [reader GetOutput]
11 eval dataMapper SetScalarRange [[reader GetOutput] \
12   GetScalarRange]

13 vtkActor dataActor
14 dataActor SetMapper dataMapper

15 vtkRenderer ren
16 ren SetBackground 1 1 1
17 ren AddActor dataActor
  
```

```

18   vtkRenderWindow renWin
19       renWin AddRenderer ren
20       renWin SetSize 300 300

21   vtkRenderWindowInteractor iren
22       iren SetRenderWindow renWin

23   iren Initialize

24   # prevent the tk window from showing up then start the event loop
25   wm withdraw .

```

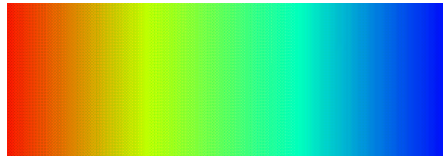


Figure 4: Colormap of a scalar field.

4 Complex data file

In subsequent examples we will use slightly more complicated data file "stripComplex.vtk". This data file defines an unstructured grid with geometry and topology exactly the same as "stripSimple.vtk" file in section 2 but with additional data set attributes. It defines new point data called "density" and cell data called "subdomains". The cell data attribute consists of integer numbers indicating the subdomain a cell belongs to. The "density" and "subdomains" attributes are shown in figure 5

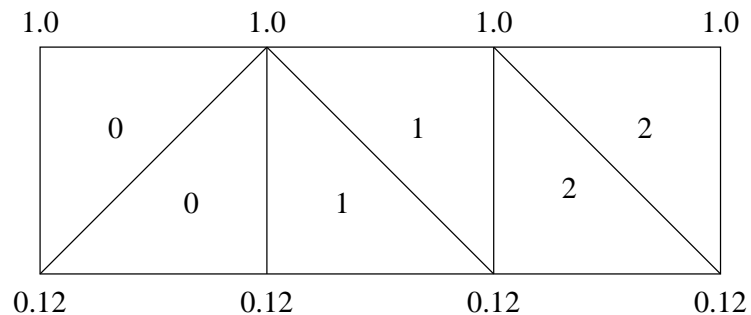


Figure 5: Point scalar attributes "temperature" and cell scalar attributes "subdomains".

The "stripComplex.vtk" file is shown below:

```
# vtk DataFile Version 3.0
```

```

2D scalar data
ASCII

DATASET UNSTRUCTURED_GRID
POINTS 8 float
0 0 2
1 0 2
1 1 2
0 1 2
2 0 2
3 0 2
3 1 2
2 1 2

CELLS 6 24
3 0 1 2
3 0 2 3
3 1 4 2
3 4 7 2
3 4 5 7
3 5 6 7

CELL_TYPES 6
5
5
5
5
5
5

POINT_DATA 8
SCALARS density float 1
LOOKUP_TABLE default
0.12 0.12 1.0 1.0
0.12 0.12 1.0 1.0

SCALARS temperature float 1
LOOKUP_TABLE default
1.0 2.0 2.0 1.0
3.0 4.0 4.0 3.0

CELL_DATA 6
SCALARS subdomains int 1
LOOKUP_TABLE default
0 0 1 1 2 2

```

5 Scalar data visualization – cont.

If we try to use script from section 3 to visualize the "temperature" field from file "stripComplex.vtk" we will note, that instead of seeing "temperature" field we see

"density" field. Closer examination of the picture reveals that there is also something wrong with colors as we do not see the whole color range. The first problem stems from the fact that `vtkUnstructuredGridReader`, unless specifically instructed, reads only the first scalar field it encounters. In file `"stripComplex.vtk"` the `"temperature"` field comes as the second and it is not read in. The second problem with colors is caused by presence of a cell scalar data attribute. As it was mentioned in section 3 the method `GetScalarRange` calculates the range for both point and cell scalar data, so the range returned by it is 0.0 to 4.0 and not as we would expect 1.0 to 4.0, for `"temperature"` field.

The script shown below is a slight modification of script from section 3 which takes into account the problems mentioned above:

```
1  package require vtk
2  package require vtkinteraction

3  # set the name of data file
4  set file "stripComplex.vtk"

5  vtkUnstructuredGridReader reader
6      reader SetFileName "$file"
7      # read "temperature" field as point data
8      reader SetScalarsName "temperature"
9      reader Update

10 # get range of scalars in points
11 set pointData [[reader GetOutput] GetPointData]
12 set drange [[${pointData} GetScalars] GetRange]

13 vtkDataSetMapper dataMapper
14     dataMapper SetInput [reader GetOutput]
15     eval dataMapper SetScalarRange $drange

16 vtkActor dataActor
17     dataActor SetMapper dataMapper

18 vtkRenderer ren
19     ren SetBackground 1 1 1
20     ren AddActor dataActor

21 vtkRenderWindow renWin
22     renWin AddRenderer ren
23     renWin SetSize 300 300

24 vtkRenderWindowInteractor iren
25     iren SetRenderWindow renWin

26 iren Initialize

27 # prevent the tk window from showing up then start the event loop
28 wm withdraw .
```

The modifications are made in two places. First in the line:

```
reader SetScalarsName "temperature"
```

we specifically instruct the grid reader to read "temperature" point data. Next in lines:

```
set pointData [[reader GetOutput] GetPointData]
set drange [[pointData GetScalars] GetRange]
```

we get the point data and calculate the range of scalars in it. That range will be exactly the range of "temperature" field.

6 Scalar data visualization with color bar

In order to provide more information about the scalar field we would like to draw a legend that relates a color to a numerical value. Such legend can be provided by `vtkColorBarActor` object, which draws a rectangular bar with colored sections, labels and title. To do its job `vtkColorBarActor` must be provided with `vtkLookupTable` object which holds the range of colors. In the script below we pass to `vtkColorBarActor` the lookup table used by the data mapper. It is possible to customize the size, position, orientation and text formatting of the color bar.

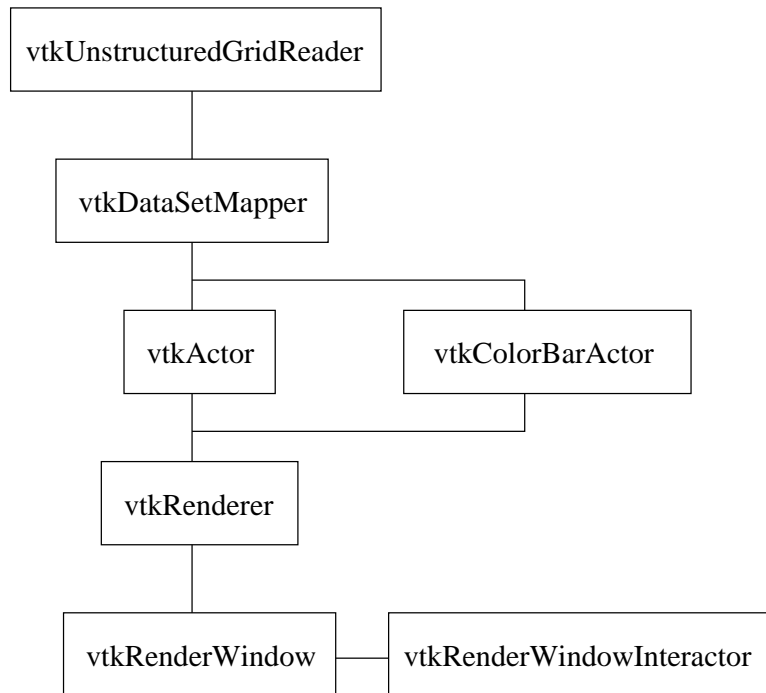


Figure 6: Colormap of a scalar field with a color bar.

```

1  package require vtk
2  package require vtkinteraction

3  # set the name of data file
4  set file "stripComplex.vtk"

5  vtkUnstructuredGridReader reader
6      reader SetFileName "$file"
7      reader SetScalarsName "temperature"
8      reader Update

9  set pointData [[reader GetOutput] GetPointData]
10 set drange [[${pointData} GetScalars] GetRange]

11 vtkDataSetMapper dataMapper
12     dataMapper SetInput [reader GetOutput]
13     eval dataMapper SetScalarRange $drange

14 vtkActor dataActor
15     dataActor SetMapper dataMapper

16 # create color legend
17 vtkScalarBarActor scalarBar
18     scalarBar SetLookupTable [dataMapper GetLookupTable]
19     scalarBar SetTitle "Temperature"

20     # configure the legend font
21     scalarBar BoldOff
22     scalarBar ShadowOff
23     [scalarBar GetProperty] SetColor 0 0 0

24     # configure position, orientation and size
25     [scalarBar GetPositionCoordinate] \
26         SetCoordinateSystemToNormalizedViewport
27     [scalarBar GetPositionCoordinate] SetValue 0.1 0.01
28     scalarBar SetOrientationToHorizontal
29     scalarBar SetWidth 0.8
30     scalarBar SetHeight 0.17

31 vtkRenderer ren
32     ren SetBackground 1 1 1
33     ren AddActor dataActor
34     ren AddActor2D scalarBar

35 vtkRenderWindow renWin
36     renWin AddRenderer ren
37     renWin SetSize 300 300

38 vtkRenderWindowInteractor iren
39     iren SetRenderWindow renWin

```

```

40   iren Initialize
41   # prevent the tk window from showing up then start the event loop
42   wm withdraw .

```

If we run the example above we can notice that red color is associated with minimum value and blue color is associated with maximum value. This is because the default lookup table defines color range from red to blue. If we want reverse mapping (i.e. blue associated with minimum value) we need to provide a custom lookup table. The code fragment below shows how to construct a `vtkLookupTable` object and set it in data mapper. The hue range in this object is set to be from 0.6667 to 0.0, which gives colors from blue to red. More details on color model used in VTK can be found in chapter 3 of [1].

```

vtkLookupTable lut
lut SetHueRange 0.6667 0.0

vtkDataSetMapper dataMapper
dataMapper SetLookupTable lut

```

The result of visualization with the scalar bar and custom lookup table is shown in figure 7.

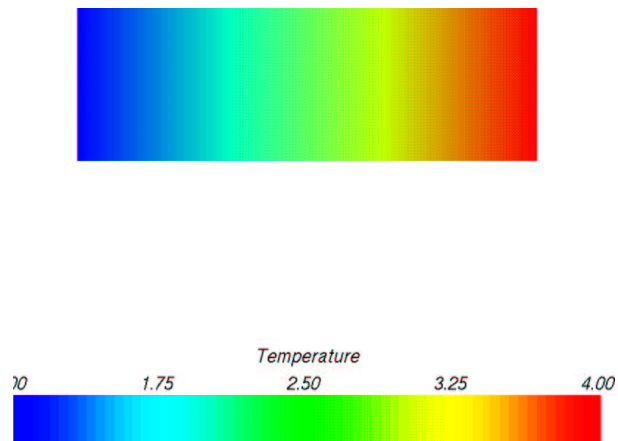


Figure 7: Colormap of a scalar field with color bar.

7 Mesh visualization

The next example introduces two filters: `vtkGeometryFilter` to obtain polygonal boundary of a grid and `vtkExtractEdges` to obtain edges of the grid cells. Instead of visualizing attributes associated with points or cells we would like to see the cells themselves. One way to do it is to draw cells in a plain color and to draw edges of cells in another color. Additionally we assume that we want to see only cells (or their faces) on the object boundary. The first filter `vtkGeometryFilter` takes as its input

an unstructured mesh and on output returns polygonal "skin" of that mesh. Then the pipeline forks.

Left branch of the pipeline is connected to `vtkPolyDataMapper`. The method `ScalarVisibilityOff` is called for this mapper so the scalar point or cell attributes will not affect a color of cells. Further, in `vtkActor` we set the color of cells and the surface properties to make all cells to appear the same, independent on their orientation.

In the right branch the geometry data is filtered through `vtkExtractEdges` filter. The output of that filter is a set of lines constituting cell edges. These edges are the input to another `vtkPolyDataMapper` where again we switch off scalar visibility. In the edges actor we set edges color to black.

This way of visualizing edges, though simple, has its drawback – with some orientation of the edges or specific picture resolution some of the edges may disappear, be only partially drawn or flicker while moving the object. A better way to draw edges is to wrap thin tubes around them using `vtkTubeFilter`. This technique is shown in section 9.

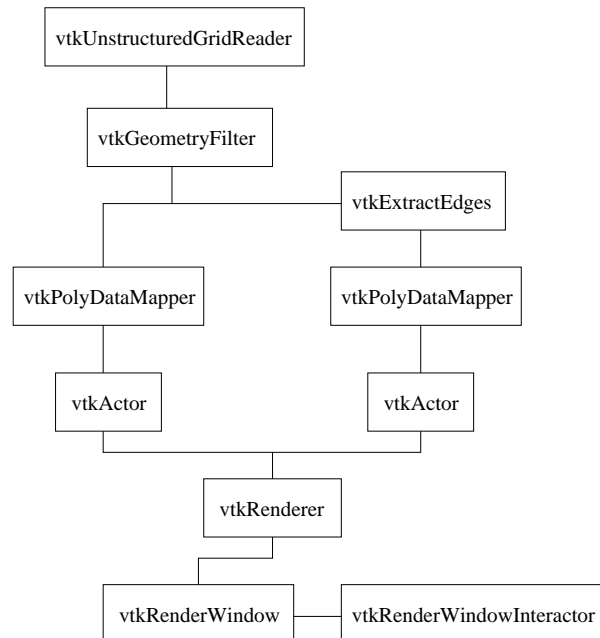


Figure 8: Mesh visualization pipeline.

```

1  package require vtk
2  package require vtkinteraction

3  # set the name of data file
4  set file "stripComplex.vtk"

5  vtkUnstructuredGridReader reader
6  reader SetFileName "$file"
  
```

```

7   # we introduce geometry filter in order to visualize only the
8   # elements (or faces of elements) on the object surface
9   vtkGeometryFilter geomFilter
10  geomFilter SetInput [reader GetOutput]

11  vtkPolyDataMapper elementsMapper
12  elementsMapper SetInput [geomFilter GetOutput]
13  # the elements will be drawn in uniform color independent on the
14  # points or cells fields
15  elementsMapper ScalarVisibilityOff

16  vtkActor elementsActor
17  elementsActor SetMapper elementsMapper

18  set elementsProp [elementsActor GetProperty]
19  # nice gold color
20  $elementsProp SetColor 0.91 0.87 0.67
21  # we switch the diffuse lighting of to make the color of the
22  # object independent on their orientation
23  $elementsProp SetDiffuse 0
24  $elementsProp SetAmbient 1
25  # we do not need any fancy shading
26  $elementsProp SetInterpolationToFlat

27  # We draw edges just as a set of lines - it is simple
28  # but has the drawback that with certain magnification and
29  # object orientation some lines might be partially obscured.
30  # The alternative would be to wrap a thin tube around
31  # each edge. This allows to draw edges as a thick 3D object
32  # which would stand out from the flat surfaces of elements.
33  vtkExtractEdges edgesFilter
34  edgesFilter SetInput [geomFilter GetOutput]

35  vtkPolyDataMapper edgesMapper
36  edgesMapper SetInput [edgesFilter GetOutput]
37  edgesMapper ScalarVisibilityOff

38  vtkActor edgesActor
39  edgesActor SetMapper edgesMapper

40  set edgesProp [edgesActor GetProperty]
41  $edgesProp SetColor 0 0 0
42  $edgesProp SetDiffuse 0
43  $edgesProp SetAmbient 1
44  $edgesProp SetLineWidth 2

45  vtkRenderer ren
46  ren SetBackground 1 1 1
47  ren AddActor elementsActor
48  ren AddActor edgesActor

```

```

49   vtkRenderWindow renWin
50       renWin AddRenderer ren
51       renWin SetSize 300 300

52   vtkRenderWindowInteractor iren
53       iren SetRenderWindow renWin

54   iren Initialize

55   # prevent the tk window from showing up then start the event loop
56   wm withdraw .

```

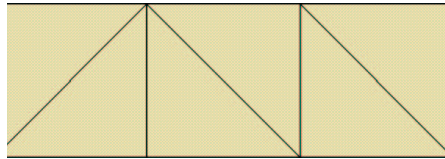


Figure 9: Visualization of cells and cell edges.

8 Subdomains visualization

This example is based on the one from previous section, however here we would like to see only elements colored according to the subdomain they belong to. As already mentioned in section 4, each cell in our data has an integer number associated with it, which indicates the subdomain where the cell belongs to. That cell data is used to gather cells belonging to one subdomain and draw them as a separate entity. Cells selection can be done using `vtkThreshold` filter. That filter takes as an input a whole mesh but as the output produces mesh which consists only of cells with data values within given range. It is possible to specify which – cell or point – data attributes should be examined, or name a particular attribute. In the example, the filter is set to look for cell attribute called "subdomains". The output of `vtkUnstructuredGridReader` forks into as many branches as the number of subdomains. In each branch a `vtkThreshold` is set in such way as to select only elements of one subdomain (it uses the fact that the subdomain numbers are integers). The output of a threshold filter is passed to a geometry filter and to a poly data mapper. In the poly data mapper we suppress scalar coloring to make all elements in one subdomain to appear the same. The color for each subdomain is assigned in its actor object. Subdomain color is specified by giving three numbers corresponding to RGB color model. The red component depends on subdomain number while the green and blue are chosen randomly.

```

1   package require vtk
2   package require vtkinteraction

3   # set the name of data file
4   set file "stripComplex.vtk"

5   vtkUnstructuredGridReader reader
6       reader SetFileName "$file"
7   # reader Update is needed because of .. GetCellData ..

```

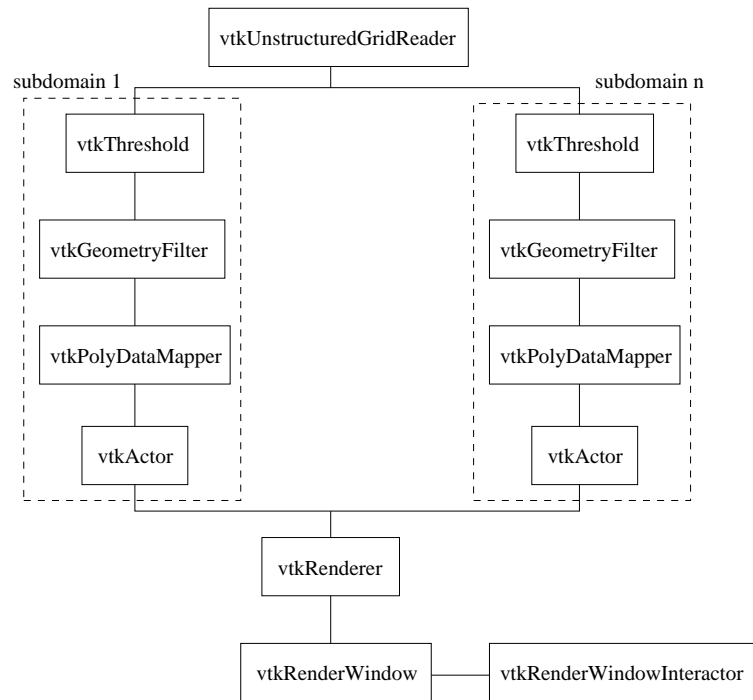


Figure 10: Subdomains visualization pipeline.

```

8      reader Update
9
10     # Get the grid and its cell data.
11     # We assume that each cell has assigned an integer value
12     # being the subdomain number. We also assume that the integers
13     # are consecutive between smin and smax. The array which
14     # hold the integers must be called "subdomains"
15     set uGrid [reader GetOutput]
16     set cellData [$uGrid GetCellData]
17
18     # set the active scalars to "subdomains" array - the cells
19     # can have more data attributes assigned to them
20     # check if "subdomains" were defined for cells
21     if {[$cellData SetActiveScalars "subdomains"] == -1} {
22       error "No subdomains data in the $file"
23       vtkCommand DeletAllObjects
24       exit
25     }
26
27     # get the "subdomains" array (it was previously set as active)
28     set subdomainArray [$cellData GetScalars]
29
30     # nice Tcl trick - GetRange return a list of two elements
31     # and we use 'foreach' command to assign the value of list
  
```



```

28 # elements to two named variables. The body of 'foreach' command
29 # is empty.
30 # The same could be done with the following way (but a bit slower
31 # if the list is long
32 # set range [$subdomainArray GetRange]
33 # set smin [lindex $range 0]
34 # set smax [lindex $range 1]

35 foreach {smin smax} [$subdomainArray GetRange] { }

36 # calculate number of subdomains
37 set nsubdomains [expr $smax - $smin + 1]

38 # set the step for the Red color component and its initial value
39 set colorstep [expr 0.9/$nsubdomains]
40 set color 0.05
41 # initialize the random number generator
42 set r [expr srand(23)]

43 vtkRenderer ren
44     ren SetBackground 1 1 1

45 # for each subdomain create its visualization pipeline
46 # each subdomain is extracted from the mesh using the
47 # vtkThreshold filter and the cell data
48 # then the subdomain geometry is extracted using vtkGeometryFilter
49 for {set i $smin} {$i <= $smax} {incr i} {
50     vtkThreshold subdomSelector$i
51         subdomSelector$i SetArrayName "subdomains"
52         subdomSelector$i SetAttributeModeToUseCellData
53         subdomSelector$i SetInput $uGrid
54         subdomSelector$i ThresholdBetween [expr $i-0.5] [expr $i+0.5]

55     vtkGeometryFilter geomFilter$i
56         geomFilter$i SetInput [subdomSelector$i GetOutput]

57     vtkPolyDataMapper subdomainMapper$i
58         subdomainMapper$i SetInput [geomFilter$i GetOutput]
59         subdomainMapper$i ScalarVisibilityOff

60     vtkActor subdomainActor$i
61         subdomainActor$i SetMapper subdomainMapper$i
62         # assign color to the subdomain actor
63         # The Green and Blue color components
64         # are taken as a random numbers
65         [subdomainActor$i GetProperty ] SetColor $color \
66             [expr rand()] [expr rand()]

67     # add the actor to the renderer
68     ren AddActor subdomainActor$i

69     # increase the value of color component

```

```

70     set color [expr $color + $colorstep]
71   }

72   vtkRenderWindow renWin
73     renWin AddRenderer ren
74     renWin SetSize 300 300

75   vtkRenderWindowInteractor iren
76     iren SetRenderWindow renWin

77   iren Initialize

78   # prevent the tk window from showing up then start the event loop
79   wm withdraw .

```

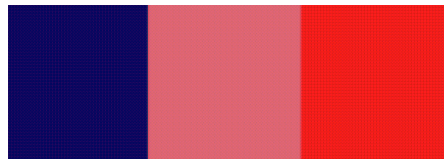


Figure 11: Visualization of subdomains.

9 Labeling points

This example is an enhancement to the example showing mesh. Besides plotting cells and their edges it also draws text labels at visible points. The labels are set to be point indices. If we plot labels at all points it could make picture totally cluttered, so instead, we plot labels only for points which fall into selected rectangular area of the plot.

The points selection area is visualized by drawing an outline of it. In the figure 12 the pipeline related to that outline is the one on the left. First, the point and cell arrays are created, and from them a `vtkPolyData` object is generated. That data object is mapped using `vtkPolyDataMapper2D`. That kind of mapper is used to draw 2D objects for which no perspective, camera and light properties are needed. Such objects are controlled by `vtkActor2D` instances.

The pipeline in the middle is the one to draw point labels. The output of `vtkUnstructuredGridReader` is forked to `vtkIdFilter`. That filter generates point attribute data (or field data) from point indices and cell attribute data (or field data) from cell indices. The next filter, `vtkSelectVisiblePoints` removes all points which do not fall into specified screen region. To draw labels we use `vtkLabeledDataMapper` which allows to specify label properties like font shape, font size, format, etc. To control labels on the screen we again use 2D actor.

```

1   package require vtk
2   package require vtkinteraction

3   # set the name of data file
4   set file "sphere.vtk"

```

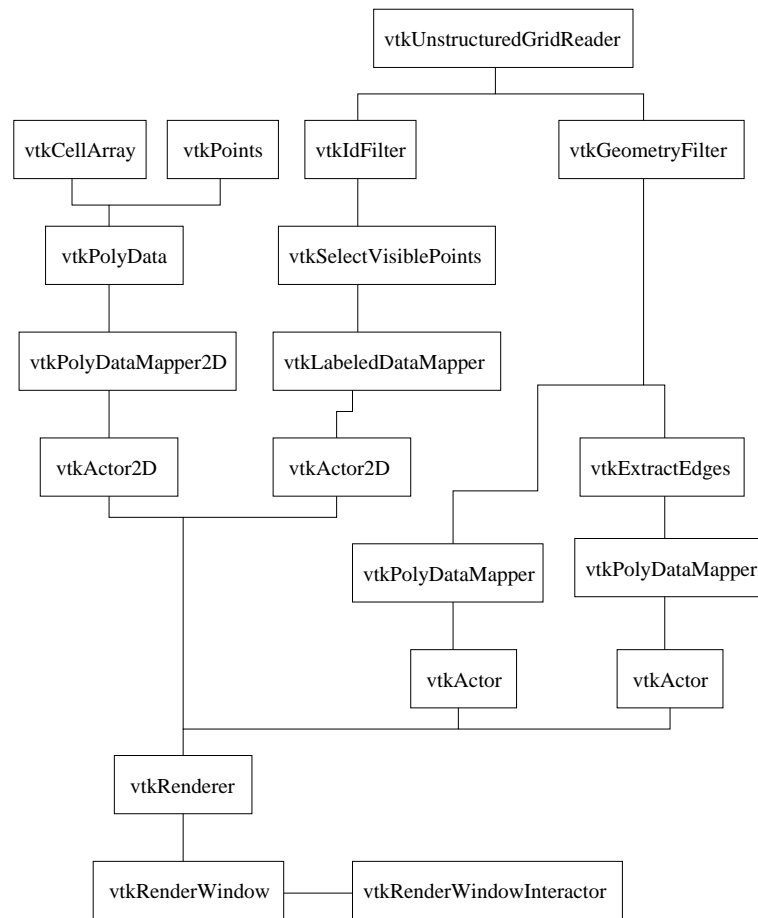


Figure 12: Labeled points visualization pipeline.

```

5   # Create selection window
6   # Only nodes which fall into this selection window will have
7   # their labels visible. This prevents to clutter the picture
8   # with to many labels
9   set xmin 100
10  set xLength 100
11  set xmax [expr $xmin + $xLength]

12  set ymin 100
13  set yLength 100
14  set ymax [expr $ymin + $yLength]

15  vtkPoints pts
16     pts InsertPoint 0 $xmin $ymin 0
17     pts InsertPoint 1 $xmax $ymin 0
18     pts InsertPoint 2 $xmax $ymax 0
  
```

```

19         pts InsertPoint 3 $xmin $ymax 0

20     vtkCellArray rect
21         rect InsertNextCell 5
22         rect InsertCellPoint 0
23         rect InsertCellPoint 1
24         rect InsertCellPoint 2
25         rect InsertCellPoint 3
26         rect InsertCellPoint 0

27     vtkPolyData selectRect
28         selectRect SetPoints pts
29         selectRect SetLines rect

30     vtkPolyDataMapper2D rectMapper
31         rectMapper SetInput selectRect

32     vtkActor2D rectActor
33         rectActor SetMapper rectMapper
34         [rectActor GetProperty] SetColor 0.5 1 0.2

35     vtkUnstructuredGridReader reader
36         reader SetFileName "$file"

37     # as in the example showMesh.tcl we will show only the elements
38     # or elements faces on the surface of the objects
39     # see the comments in showMesh.tcl
40     vtkGeometryFilter geomFilter
41         geomFilter SetInput [reader GetOutput]

42     vtkPolyDataMapper elementsMapper
43         elementsMapper SetInput [geomFilter GetOutput]
44         elementsMapper ScalarVisibilityOff

45     vtkActor elementsActor
46         elementsActor SetMapper elementsMapper

47     set elementsProp [elementsActor GetProperty]
48         $elementsProp SetColor 0.91 0.87 0.67
49     # $elementsProp SetColor 1 1 1
50     $elementsProp SetDiffuse 0
51     $elementsProp SetAmbient 1
52     $elementsProp SetInterpolationToFlat

53     vtkExtractEdges edgesFilter
54         edgesFilter SetInput [geomFilter GetOutput]

55     vtkPolyDataMapper edgesMapper
56         edgesMapper SetInput [edgesFilter GetOutput]
57         edgesMapper ScalarVisibilityOff

58     vtkActor edgesActor

```

```

59         edgesActor SetMapper edgesMapper

60     set edgesProp [edgesActor GetProperty]
61         $edgesProp SetColor 0 0 0
62         $edgesProp SetDiffuse 0
63         $edgesProp SetAmbient 1
64         $edgesProp SetLineWidth 2

65     # Generate data arrays containing points ids
66     vtkIdFilter ids
67         ids SetInput [reader GetOutput]
68         # we select only ids for points
69         ids PointIdsOn

70     # we need to create renderer here because vtkSelectVisiblePoints
71     # needs it
72     vtkRenderer ren
73         ren SetBackground 1 1 1

74     # Create labels for points
75     vtkSelectVisiblePoints visPts
76         visPts SetInput [ids GetOutput]
77         visPts SetRenderer ren
78         visPts SelectionWindowOn
79         visPts SetSelection $xmin $xmax $ymin $ymax

80     # The mapper which draw the labels on data set points
81     vtkLabeledDataMapper plm
82         plm SetInput [visPts GetOutput]
83         plm SetLabelFormat "%g"
84         plm ShadowOff
85         plm SetFontSize 14
86         plm SetLabelModeToLabelFieldData
87         plm BoldOff

88     vtkActor2D pointLabels
89         pointLabels SetMapper plm
90         eval [pointLabels GetProperty] SetColor 0 0 1

91     ren AddActor elementsActor
92     ren AddActor edgesActor
93     ren AddActor rectActor
94     ren AddActor pointLabels

95     vtkRenderWindow renWin
96         renWin AddRenderer ren
97         renWin SetSize 300 300

98     vtkRenderWindowInteractor iren
99         iren SetRenderWindow renWin

100    iren Initialize

```

```

101 # prevent the tk window from showing up then start the event loop
102 wm withdraw .

```

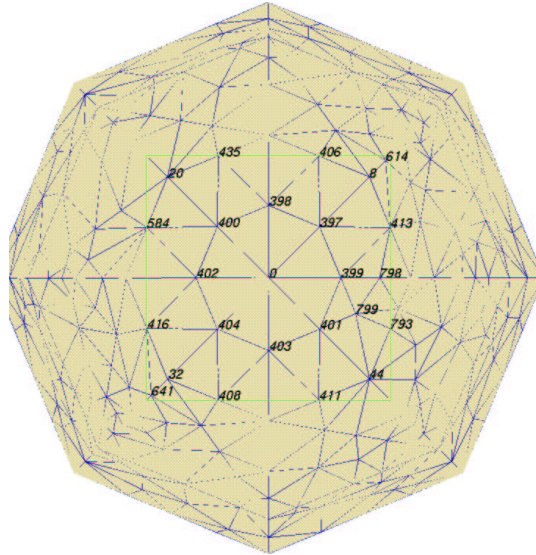


Figure 13: Visualization of mesh with points labeled by their index.

Figure 13 shows sphere mesh with point labels. It can be seen that some of the edges are not well visible. We can improve the visualization by applying `vtkTubeFilter`. The output of `vtkExtractEdges` is filtered through `vtkTubeFilter` and only then send to `vtkPolyDataMapper`. The pipeline modification is shown in figure 14 and the corresponding code is given below:

```

vtkExtractEdges edgesFilter
edgesFilter SetInput [geomFilter GetOutput]

vtkTubeFilter tubeFilter
tubeFilter SetInput [edgesFilter GetOutput]
tubeFilter SetRadius 0.05

vtkPolyDataMapper edgesMapper
edgesMapper SetInput [tubeFilter GetOutput]
edgesMapper ScalarVisibilityOff

```

The result of running the improved script is shown in figure 15.

10 Labeling cells

This example is almost a copy of the previous one except that here only cells are labeled. The single addition to the scheme in figure 12 is the filter `vtkCellCenters` which on input takes any data set and on output generates set of points in the cell

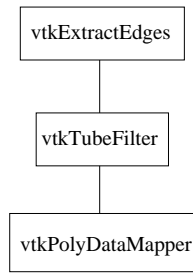


Figure 14: Labeled points visualization pipeline.

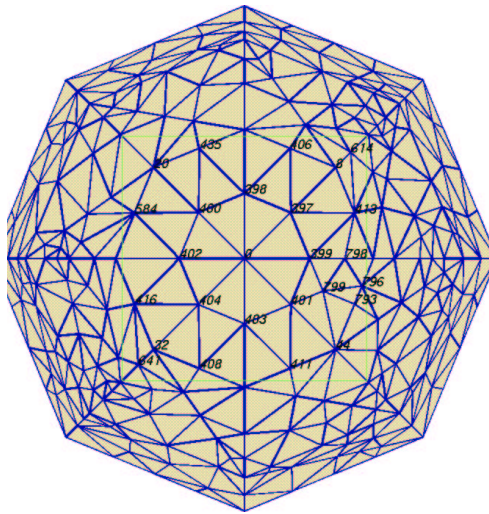


Figure 15: Visualization of mesh with edges represented by tubes.

centers. These points are then passed to `vtkSelectVisiblePoints` filter like in the previous example.

The figures 17 and 18 are the visualization results with and without activating `vtkSelectVisiblePoints`, respectively. As it was already mentioned restricting number of labels make the picture more readable. There is however a drawback of using `vtkSelectVisiblePoints` filter. If the mesh contains volume elements (tetrahedra or cubes) then the cell centers are not visible in any circumstances and their labels are not printed. Switching point selection off (by using the method `SelectionWindowOff`) fixes this problem but it make sense only for meshes with small number of cells, otherwise the picture can be totally obscured by overlapping labels.

```

1  package require vtk
2  package require vtkinteraction

3  # set the name of data file
4  set file "stripComplex.vtk"
  
```

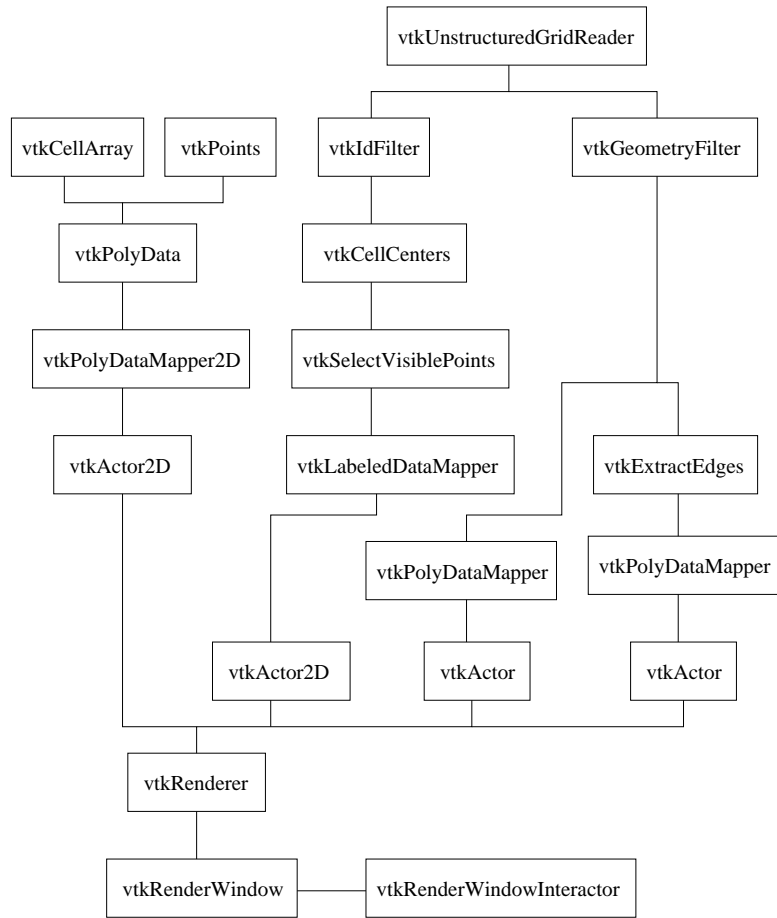


Figure 16: Labeled cells visualization pipeline.

```

5   # Create selection window
6   # Only cell centers which fall into this selection window
7   # will have their labels visible. This prevents to clutter
8   # the picture with too many labels.
9   set xmin 100
10  set xLength 100
11  set xmax [expr $xmin + $xLength]

12  set ymin 100
13  set yLength 100
14  set ymax [expr $ymin + $yLength]

15  vtkPoints pts
16     pts InsertPoint 0 $xmin $ymin 0
17     pts InsertPoint 1 $xmax $ymin 0
18     pts InsertPoint 2 $xmax $ymax 0
19     pts InsertPoint 3 $xmin $ymax 0
  
```



```

20  vtkCellArray rect
21      rect InsertNextCell 5
22      rect InsertCellPoint 0
23      rect InsertCellPoint 1
24      rect InsertCellPoint 2
25      rect InsertCellPoint 3
26      rect InsertCellPoint 0

27  vtkPolyData selectRect
28      selectRect SetPoints pts
29      selectRect SetLines rect

30  vtkPolyDataMapper2D rectMapper
31      rectMapper SetInput selectRect

32  vtkActor2D rectActor
33      rectActor SetMapper rectMapper
34      [rectActor GetProperty] SetColor 0.5 1 0.2

35  vtkUnstructuredGridReader reader
36      reader SetFileName "$file"

37  # as in the example showMesh.tcl we will show only the elements
38  # or elements faces on the surface of the objects
39  # see the comments in showMesh.tcl

40  vtkGeometryFilter geomFilter
41      geomFilter SetInput [reader GetOutput]

42  vtkPolyDataMapper elementsMapper
43      elementsMapper SetInput [geomFilter GetOutput]
44      elementsMapper ScalarVisibilityOff

45  vtkActor elementsActor
46      elementsActor SetMapper elementsMapper

47  set elementsProp [elementsActor GetProperty]
48      $elementsProp SetColor 0.91 0.87 0.67
49      $elementsProp SetDiffuse 0
50      $elementsProp SetAmbient 1
51      $elementsProp SetInterpolationToFlat

52  vtkExtractEdges edgesFilter
53      edgesFilter SetInput [geomFilter GetOutput]

54  vtkPolyDataMapper edgesMapper
55      edgesMapper SetInput [edgesFilter GetOutput]
56      edgesMapper ScalarVisibilityOff

57  vtkActor edgesActor
58      edgesActor SetMapper edgesMapper

```

```

59 set edgesProp [edgesActor GetProperty]
60     $edgesProp SetColor 0 0 0
61     $edgesProp SetDiffuse 0
62     $edgesProp SetAmbient 1
63     $edgesProp SetLineWidth 2

64 # Generate data arrays containing cells ids
65 vtkIdFilter ids
66     ids SetInput [reader GetOutput]
67     # select only cell ids
68     ids CellIdsOn

69 # we need to create renderer here because vtkSelectVisiblePoints
70 vtkRenderer ren
71     ren SetBackground 1 1 1

72 # Find the centers of the cells specified by the array ids
73 vtkCellCenters cc
74     cc SetInput [ids GetOutput]

75 # Create labels for cells
76 vtkSelectVisiblePoints visCells
77     visCells SetInput [cc GetOutput]
78     visCells SetRenderer ren
79     visCells SetSelection $xmin $xmax $ymin $ymax
80 # You can excersise difference of using switching
81 # selection on and off.
82 # Switching selection off makes all cell numbers visible
83 # regardless of the fact that if they into selection
84 # frame or are obscured by other elements or are inside volumes
85     visCells SelectionWindowOn

86 vtkLabeledDataMapper clm
87     clm SetInput [visCells GetOutput]
88     clm SetLabelFormat "%g"
89     clm ShadowOff
90     clm SetFontSize 20
91     clm SetLabelModeToLabelFieldData
92     clm BoldOff

93 vtkActor2D cellLabels
94     cellLabels SetMapper clm
95     eval [cellLabels GetProperty] SetColor 0 0 0

96 ren AddActor elementsActor
97 ren AddActor edgesActor
98 ren AddActor rectActor
99 ren AddActor cellLabels

100 vtkRenderWindow renWin
101     renWin AddRenderer ren

```

```

102     renWin SetSize 300 300

103     vtkRenderWindowInteractor iren
104     iren SetRenderWindow renWin

105     iren Initialize

106     # prevent the tk window from showing up then start the event loop
107     wm withdraw .

```

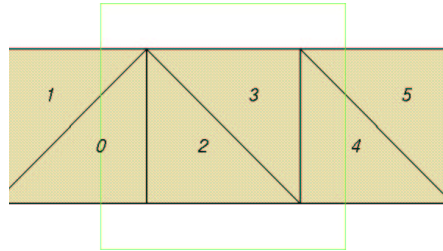


Figure 17: Visualization of a mesh with cells labeled by their index. All cells are labeled.

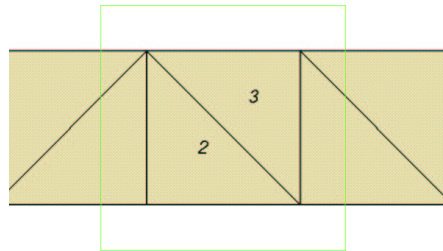


Figure 18: Visualization of a mesh with cells labeled by their index. Only selected cells are labeled.

11 Saving visualization to a file

The final result of visualization process, is an image which can be either displayed in a window or saved to a file. The simplest way to save the result of a visualization is to use the display window as the source of image. It can be done using `vtkWindowToImageFilter`, which takes a window as an input and on output produces an image data. Then the image can be saved to a file. The drawback of the presented solution is that the rendering window must be raised and fully visible, otherwise the picture will be obscured by parts of other windows. The more advanced solution would be to use off-screen rendering, where an image is rendered to a window kept in memory.

Below we show procedure `ExportPS`, which can be called to produce postscript output of a window image. The name of the output file is derived from the base name of input file, plus `.ps` extension.

```

proc ExportPS type {

```

```

# get file name from the reader
set filename [reader GetFileName]
# substitute or add .ps extension
set filename "[file root $filename].ps"

vtkPostScriptWriter writerPS
vtkWindowToImageFilter w2if
w2if SetInput renWin
writerPS SetInput [renderer GetOutput]
writerPS SetFileName $filename
writerPS Write
writerPS Delete
w2if Delete
}

```

One way of calling the above procedure is to bind it with user event handler in `vtkRenderWindowInteractor`. Each VTK class provides a method `AddObserver`. Adding observer means that it will look at the events invoked on the object (e.g. mouse click or key press) and if encounters an event it is supposed to handle it will call associated function. The class `vtkRenderWindowInteractor` can have an observer for `UserEvent` which is initiated by user pressing the 'u' key. The code below adds an observer to window interactor `iren` and sets as its callback the `ExportPS` command shown above.

```

iren AddObserver UserEvent {ExportPS}

```

The complete part of the script to save the visualization result as a postscript file is :

```

vtkRenderWindowInteractor iren
iren SetRenderWindow renWin
iren AddObserver UserEvent {ExportPS}

```

References

- [1] W. Schroeder, K. Martin, B. Lorensen: The Visualization Toolkit An Object-Oriented Approach To 3D Graphics, 3rd Edition. Kitware, Inc. 2003
- [2] The VTK User's Guide. Edited by W. Shroeder. Kitware, Inc. 2003
- [3] B.B. Welch: Practical programming in Tcl and Tk. 3rd edition, Prentice-Hall, 1999.
- [4] J.K. Ousterhout: Tcl and the Tk Toolkit. Professional Computing Series. Addison-Wesley, 1994.