Simple visualizations of unstructured grids with VTK

Roman Putanowicz, Frédéric Magoulès

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

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 `vtkPolyDataMaper`, 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 script[1] 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

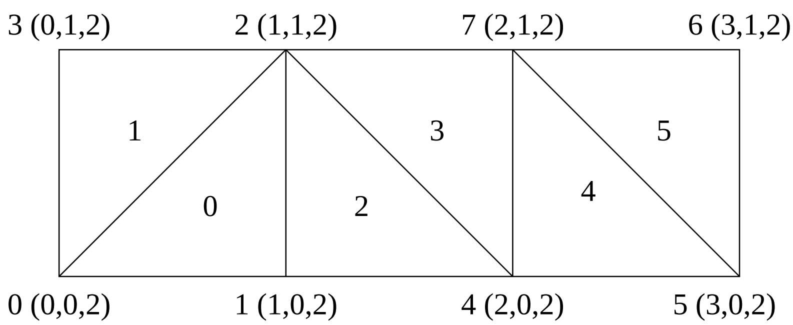

Figure 1: Nodes and elements numbering. Node coordinates are shown in brackets.

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.

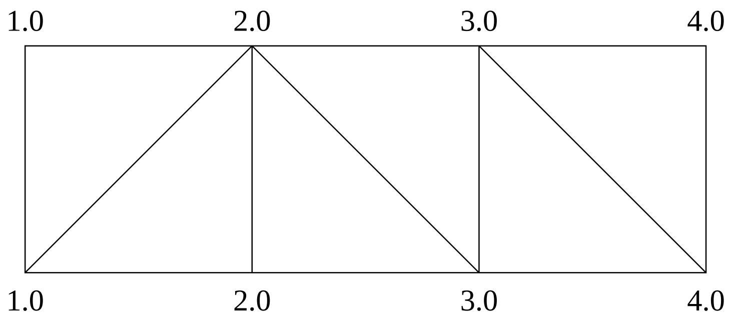

Figure 2: Point attributes `"temperature"`.

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 `vktUnstructuredGridReader`.

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.

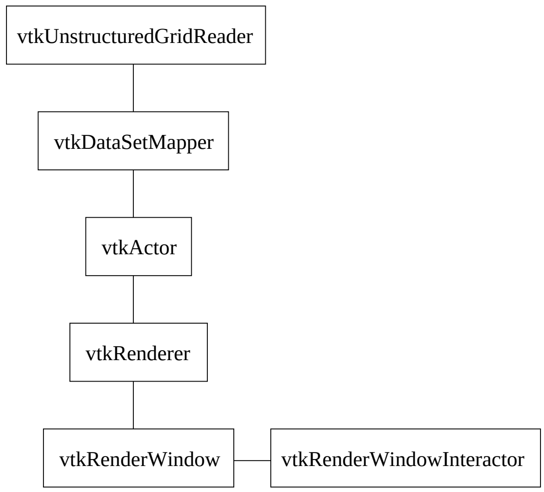

Figure 3: Scalar data visualization pipeline.

```
package require vtk
package require vtkinteraction

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

vtkUnstructuredGridReader reader
    reader SetFileName "$file"
# reader Update is needed because of .. GetScalarRange ..
    reader Update

vtkDataSetMapper dataMapper
    dataMapper SetInput [reader GetOutput]
    eval dataMapper SetScalarRange [[reader GetOutput] \
             GetScalarRange]

vtkActor dataActor
    dataActor SetMapper dataMapper

vtkRenderer ren
    ren SetBackground 1 1 1
    ren AddActor dataActor
```

```
vtkRenderWindow renWin
    renWin AddRenderer ren
    renWin SetSize 300 300

vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

iren Initialize

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

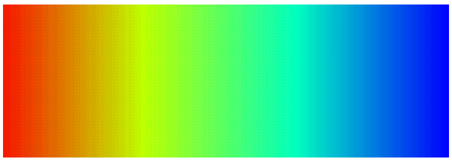

Figure 4: Colormap of a scalar field.

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

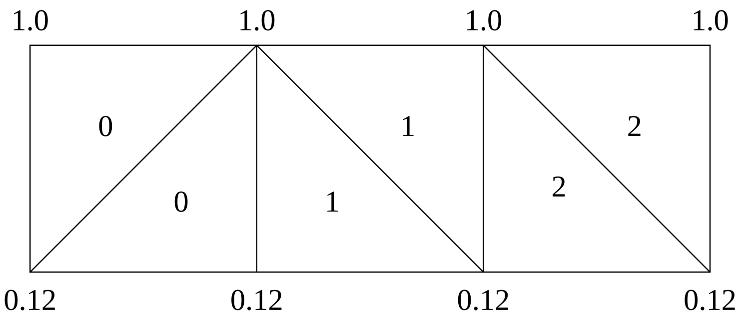

Figure 5: Point scalar attributes "temperature" and cell scalar attributes "subdomains".

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 steams 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:

```
package require vtk
package require vtkinteraction

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

vtkUnstructuredGridReader reader
    reader SetFileName "$file"
    # read "temperature" field as point data
    reader SetScalarsName "temperature"
    reader Update

# get range of scalars in points
set pointData [[reader GetOutput] GetPointData]
set drange [[$pointData GetScalars] GetRange]

vtkDataSetMapper dataMapper
    dataMapper SetInput [reader GetOutput]
    eval dataMapper SetScalarRange $drange

vtkActor dataActor
    dataActor SetMapper dataMapper

vtkRenderer ren
  ren SetBackground 1 1 1
  ren AddActor dataActor

vtkRenderWindow renWin
  renWin AddRenderer ren
  renWin SetSize 300 300

vtkRenderWindowInteractor iren
  iren SetRenderWindow renWin

iren Initialize

# prevent the tk window from showing up then start the event loop
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 formating of the color bar.

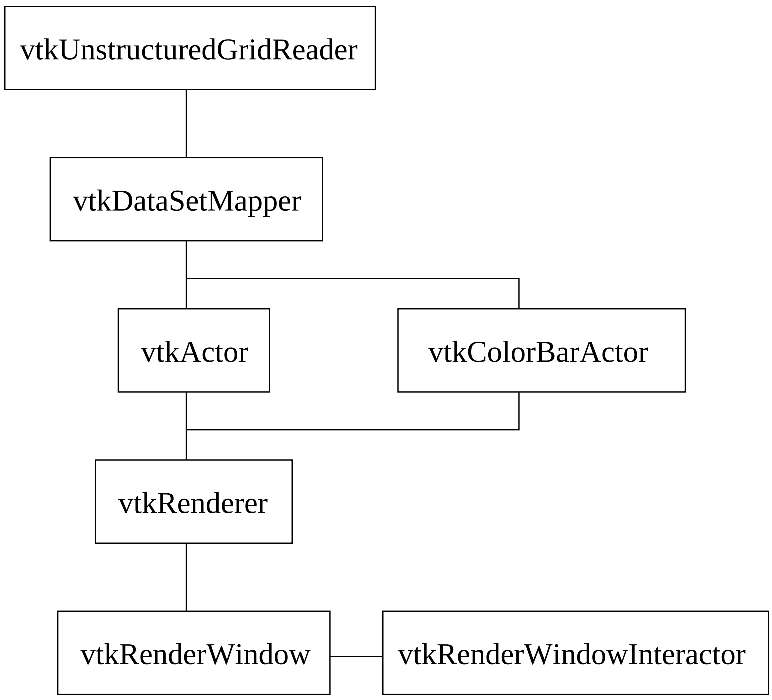

Figure 6: Colormap of a scalar field with a color bar.

```
package require vtk
package require vtkinteraction

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

vtkUnstructuredGridReader reader
    reader SetFileName "$file"
    reader SetScalarsName "temperature"
    reader Update

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

vtkDataSetMapper dataMapper
    dataMapper SetInput [reader GetOutput]
    eval dataMapper SetScalarRange $drange

vtkActor dataActor
    dataActor SetMapper dataMapper

# create color legend
vtkScalarBarActor scalarBar
    scalarBar SetLookupTable [dataMapper GetLookupTable]
    scalarBar SetTitle "Temperature"

    # configure the legend font
    scalarBar BoldOff
    scalarBar ShadowOff
    [scalarBar GetProperty] SetColor 0 0 0

    # configure position, orientation and size
    [scalarBar GetPositionCoordinate] \
                     SetCoordinateSystemToNormalizedViewport
    [scalarBar GetPositionCoordinate] SetValue 0.1 0.01
    scalarBar SetOrientationToHorizontal
    scalarBar SetWidth 0.8
    scalarBar SetHeight 0.17

vtkRenderer ren
  ren SetBackground 1 1 1
  ren AddActor dataActor
  ren AddActor2D scalarBar

vtkRenderWindow renWin
  renWin AddRenderer ren
  renWin SetSize 300 300

vtkRenderWindowInteractor iren
  iren SetRenderWindow renWin
```

```
iren Initialize

# prevent the tk window from showing up then start the event loop
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 form 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.

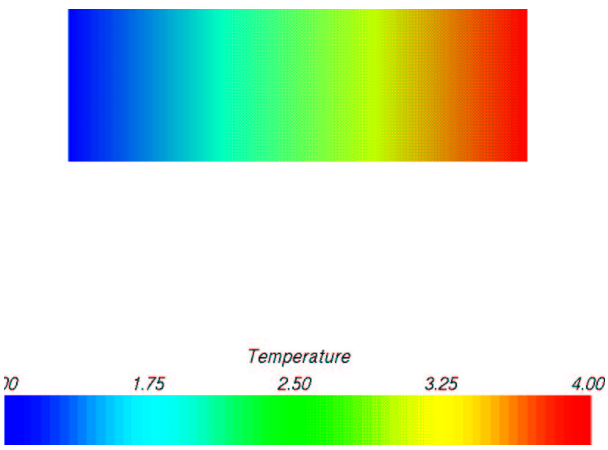

Figure 7: Colormap of a scalar field with color bar.

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 `vktPolyDataMapper` 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.

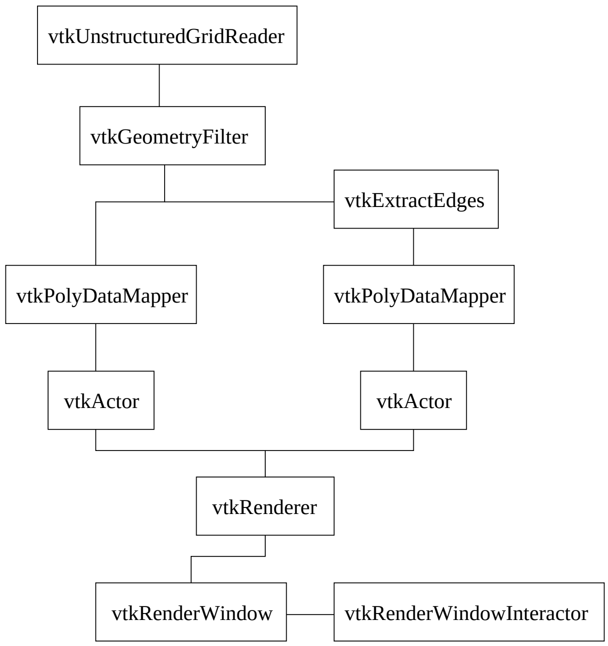

Figure 8: Mesh visualization pipeline.

```
package require vtk
package require vtkinteraction

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

vtkUnstructuredGridReader reader
    reader SetFileName "$file"
```

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

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

vtkActor elementsActor
    elementsActor SetMapper elementsMapper

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

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

vtkPolyDataMapper edgesMapper
    edgesMapper SetInput [edgesFilter GetOutput]
    edgesMapper ScalarVisibilityOff

vtkActor edgesActor
    edgesActor SetMapper edgesMapper

set edgesProp [edgesActor GetProperty]
    $edgesProp SetColor 0 0 0
    $edgesProp SetDiffuse 0
    $edgesProp SetAmbient 1
    $edgesProp SetLineWidth 2

vtkRenderer ren
    ren SetBackground 1 1 1
    ren AddActor elementsActor
    ren AddActor edgesActor
```

```
vtkRenderWindow renWin
    renWin AddRenderer ren
    renWin SetSize 300 300

vtkRenderWindowInteractor iren
   iren SetRenderWindow renWin

iren Initialize

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

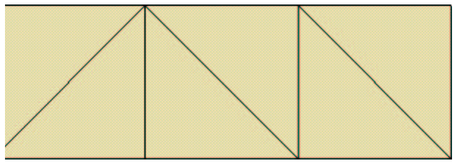

Figure 9: Visualization of cells and cell edges.

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.

```
package require vtk
package require vtkinteraction

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

vtkUnstructuredGridReader reader
    reader SetFileName "$file"
# reader Update is needed because of .. GetCellData ..
```

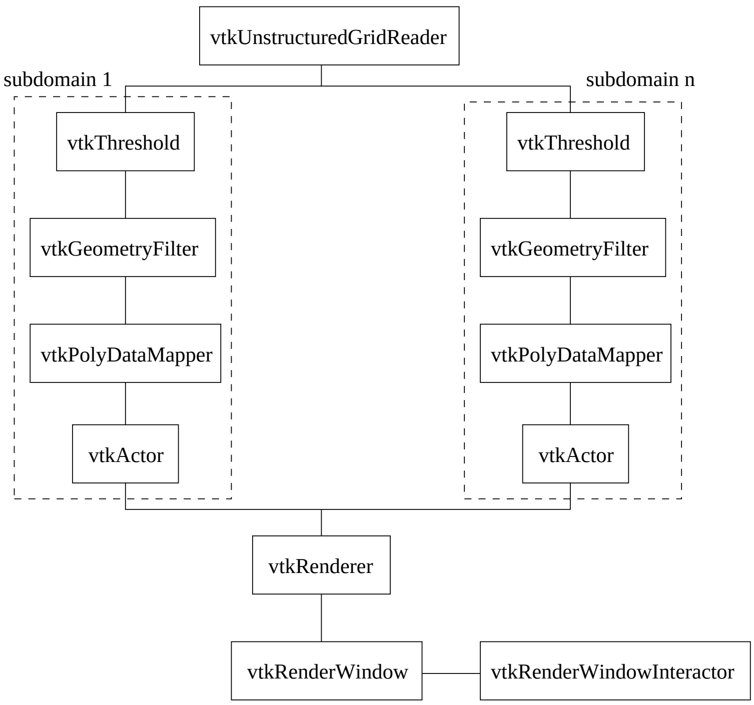

Figure 10: Subomains visualization pipeline.

```
    reader Update

# Get the grid and its cell data.
# We assume that each cell has assigned an integer value
# being the subdomain number. We also assume that the integers
# are consecutive between smin and smax. The array which
# hold the integers must be called "subdomains"
set uGrid [reader GetOutput]
set cellData [$uGrid GetCellData]

# set the active scalars to "subdomains" array - the cells
# can have more data attributes assigned to them
# check if "subdomains" were defined for cells
if {[$cellData SetActiveScalars "subdomains"] == -1} {
   error "No subdomains data in the $file"
   vtkCommand DeletAllObjects
   exit
}

# get the "subdomains" array (it was previously set as active)
set subdomainArray [$cellData GetScalars]

# nice Tcl trick - GetRange return a list of two elements
# and we use 'foreach' command to assign the value of list
```

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

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

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

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

vtkRenderer ren
   ren SetBackground 1 1 1

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

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

   vtkPolyDataMapper subdomainMapper$i
      subdomainMapper$i SetInput [geomFilter$i GetOutput]
      subdomainMapper$i ScalarVisibilityOff

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

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

   # increase the value of color component
```

```
   set color [expr $color + $colorstep]
}

vtkRenderWindow renWin
   renWin AddRenderer ren
   renWin SetSize 300 300

vtkRenderWindowInteractor iren
   iren SetRenderWindow renWin

iren Initialize

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

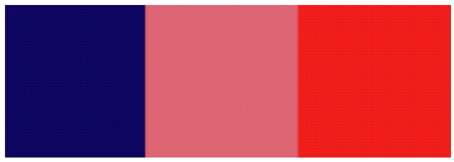

Figure 11: Visualization of subdomains.

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 `vktPolyData` 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 `vktLabeledDataMapper` which allows to specify label properties like font shape, font size, format, etc. To control labels on the screen we again use 2D actor.

```
package require vtk
package require vtkinteraction

# set the name of data file
set file "sphere.vtk"
```

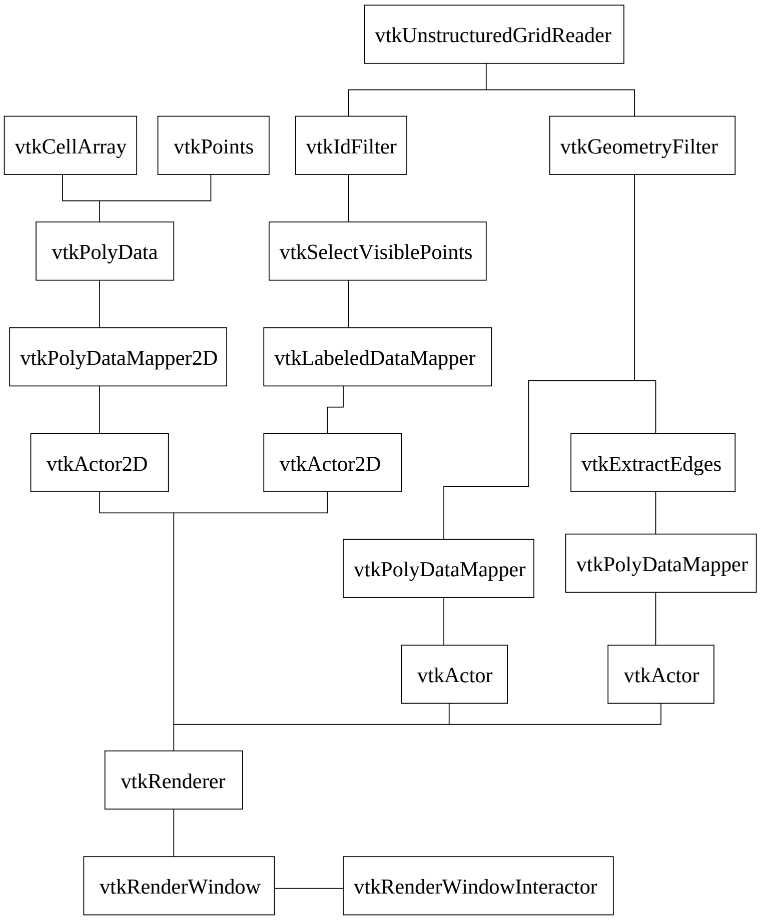

Figure 12: Labeled points visualization pipeline.

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

set ymin 100
set yLength 100
set ymax [expr $ymin + $yLength]

vtkPoints pts
    pts InsertPoint 0 $xmin $ymin 0
    pts InsertPoint 1 $xmax $ymin 0
    pts InsertPoint 2 $xmax $ymax 0
```

```
    pts InsertPoint 3 $xmin $ymax 0

vtkCellArray rect
    rect InsertNextCell 5
    rect InsertCellPoint 0
    rect InsertCellPoint 1
    rect InsertCellPoint 2
    rect InsertCellPoint 3
    rect InsertCellPoint 0

vtkPolyData selectRect
    selectRect SetPoints pts
    selectRect SetLines rect

vtkPolyDataMapper2D rectMapper
    rectMapper SetInput selectRect

vtkActor2D rectActor
    rectActor SetMapper rectMapper
    [rectActor GetProperty] SetColor 0.5 1 0.2

vtkUnstructuredGridReader reader
    reader SetFileName "$file"

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

vtkPolyDataMapper elementsMapper
    elementsMapper SetInput [geomFilter GetOutput]
    elementsMapper ScalarVisibilityOff

vtkActor elementsActor
    elementsActor SetMapper elementsMapper

set elementsProp [elementsActor GetProperty]
    $elementsProp SetColor 0.91 0.87 0.67
#   $elementsProp SetColor  1    1    1
    $elementsProp SetDiffuse 0
    $elementsProp SetAmbient 1
    $elementsProp SetInterpolationToFlat

vtkExtractEdges edgesFilter
    edgesFilter SetInput [geomFilter GetOutput]

vtkPolyDataMapper edgesMapper
    edgesMapper SetInput [edgesFilter GetOutput]
    edgesMapper ScalarVisibilityOff

vtkActor edgesActor
```

```
    edgesActor SetMapper edgesMapper
set edgesProp [edgesActor GetProperty]
    $edgesProp SetColor 0 0 0
    $edgesProp SetDiffuse 0
    $edgesProp SetAmbient 1
    $edgesProp SetLineWidth 2
# Generate data arrays containing points ids
vtkIdFilter ids
    ids SetInput [reader GetOutput]
    # we select only ids for points
    ids PointIdsOn
# we need to create renderer here because vtkSelectVisiblePoints
# needs it
vtkRenderer ren
    ren SetBackground 1 1 1
# Create labels for points
vtkSelectVisiblePoints visPts
    visPts SetInput [ids GetOutput]
    visPts SetRenderer ren
    visPts SelectionWindowOn
    visPts SetSelection $xmin $xmax $ymin $ymax
# The mapper which draw the labels on data set poins
vtkLabeledDataMapper plm
    plm SetInput [visPts GetOutput]
    plm SetLabelFormat "%g"
    plm ShadowOff
    plm SetFontSize 14
    plm SetLabelModeToLabelFieldData
    plm BoldOff
vtkActor2D pointLabels
    pointLabels SetMapper plm
    eval [pointLabels GetProperty] SetColor 0 0 1
ren AddActor elementsActor
ren AddActor edgesActor
ren AddActor rectActor
ren AddActor pointLabels
vtkRenderWindow renWin
   renWin AddRenderer ren
   renWin SetSize 300 300
vtkRenderWindowInteractor iren
  iren SetRenderWindow renWin
iren Initialize
```

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

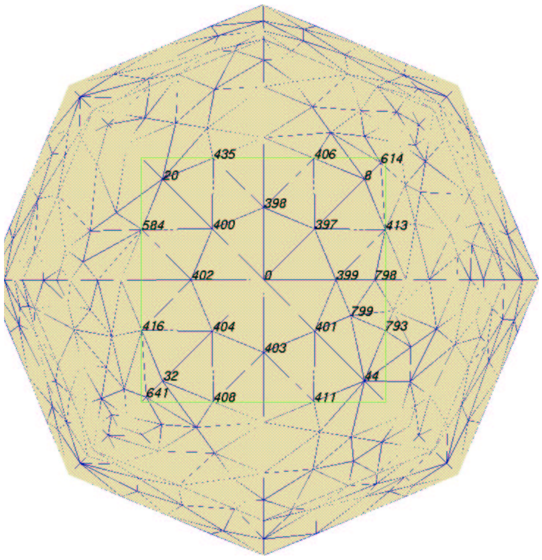

Figure 13: Visualization of mesh with points labeled by their index.

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

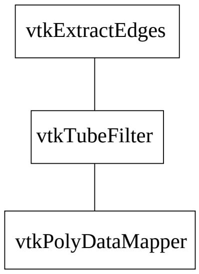

Figure 14: Labeled points visualization pipeline.

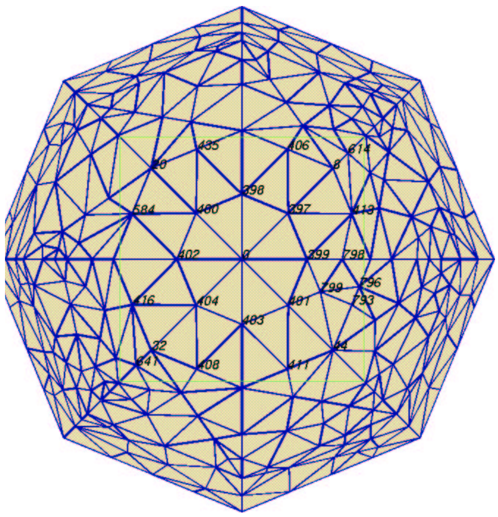

Figure 15: Visualization of mesh with edges represented by tubes.

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 `vktSelectVisiblePoints` 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.

```
package require vtk
package require vtkinteraction

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

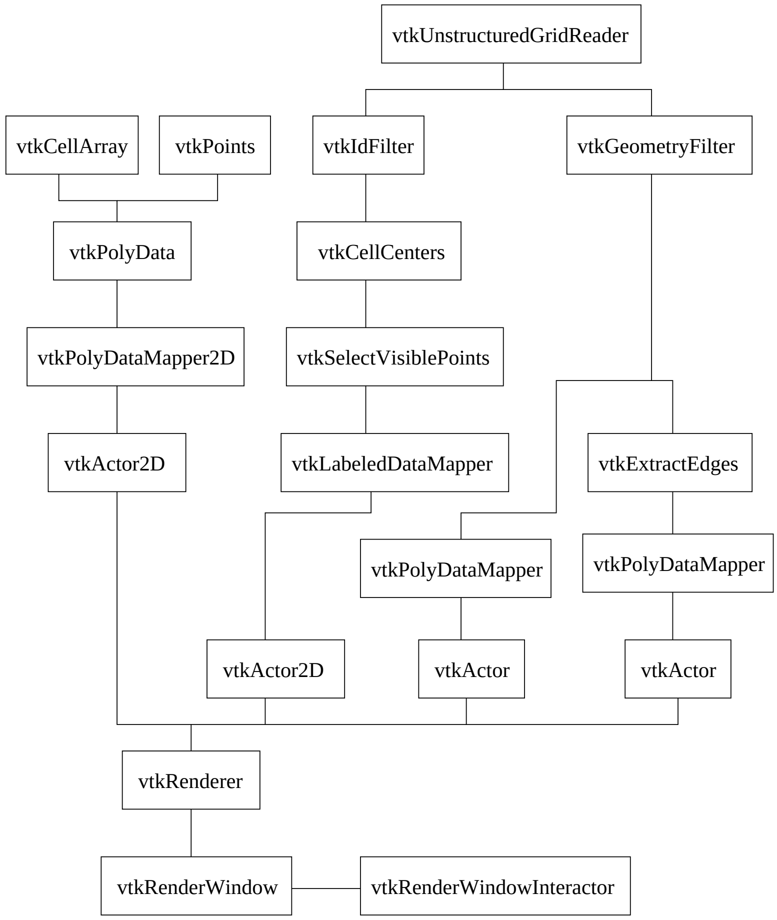

Figure 16: Labeled cells visualization pipeline.

```
# Create selection window
# Only cell centers which fall into this selection window
# will have their labels visible. This prevents to clutter
# the picture with to many labels.
set xmin 100
set xLength 100
set xmax [expr $xmin + $xLength]

set ymin 100
set yLength 100
set ymax [expr $ymin + $yLength]

vtkPoints pts
    pts InsertPoint 0 $xmin $ymin 0
    pts InsertPoint 1 $xmax $ymin 0
    pts InsertPoint 2 $xmax $ymax 0
    pts InsertPoint 3 $xmin $ymax 0
```

```
vtkCellArray rect
    rect InsertNextCell 5
    rect InsertCellPoint 0
    rect InsertCellPoint 1
    rect InsertCellPoint 2
    rect InsertCellPoint 3
    rect InsertCellPoint 0

vtkPolyData selectRect
    selectRect SetPoints pts
    selectRect SetLines rect

vtkPolyDataMapper2D rectMapper
    rectMapper SetInput selectRect

vtkActor2D rectActor
    rectActor SetMapper rectMapper
    [rectActor GetProperty] SetColor 0.5 1 0.2

vtkUnstructuredGridReader reader
    reader SetFileName "$file"

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

vtkGeometryFilter geomFilter
    geomFilter SetInput [reader GetOutput]

vtkPolyDataMapper elementsMapper
    elementsMapper SetInput [geomFilter GetOutput]
    elementsMapper ScalarVisibilityOff

vtkActor elementsActor
    elementsActor SetMapper elementsMapper

set elementsProp [elementsActor GetProperty]
    $elementsProp SetColor 0.91 0.87 0.67
    $elementsProp SetDiffuse 0
    $elementsProp SetAmbient 1
    $elementsProp SetInterpolationToFlat

vtkExtractEdges edgesFilter
    edgesFilter SetInput [geomFilter GetOutput]

vtkPolyDataMapper edgesMapper
    edgesMapper SetInput [edgesFilter GetOutput]
    edgesMapper ScalarVisibilityOff

vtkActor edgesActor
    edgesActor SetMapper edgesMapper
```

```
set edgesProp [edgesActor GetProperty]
    $edgesProp SetColor 0 0 0
    $edgesProp SetDiffuse 0
    $edgesProp SetAmbient 1
    $edgesProp SetLineWidth 2

# Generate data arrays containing cells ids
vtkIdFilter ids
    ids SetInput [reader GetOutput]
    # select only cell ids
    ids CellIdsOn

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

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

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

vtkLabeledDataMapper clm
    clm SetInput [visCells GetOutput]
    clm SetLabelFormat "%g"
    clm ShadowOff
    clm SetFontSize 20
    clm SetLabelModeToLabelFieldData
    clm BoldOff

vtkActor2D cellLabels
    cellLabels SetMapper clm
    eval [cellLabels GetProperty] SetColor 0 0 0

ren AddActor elementsActor
ren AddActor edgesActor
ren AddActor rectActor
ren AddActor cellLabels

vtkRenderWindow renWin
   renWin AddRenderer ren
```

```
    renWin SetSize 300 300

vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

iren Initialize

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

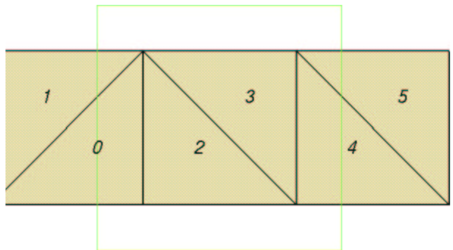

Figure 17: Visualization of a mesh with cells labeled by their index. All cells are labeled.

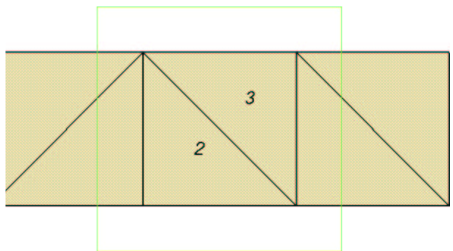

Figure 18: Visualization of a mesh with cells labeled by their index. Only selected cells are labeled.

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 `UserEven` 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.