Assignment 02: Scalar Field Visualization with Paraview

Scott Merkley

February/07/2023

Department of Computer Science, University of Utah, Salt Lake City, 84112, UT, USA

Introduction

In this report I will be explaining the results and figures from the questions asked in Assignment 02 for CS 5635 at the University of Utah. This assignment overviews visualization of 2D images, contour plotting, visualization of 3D images, and coding with Python script while using Paraview.

Part 1: Load the Data

Q1: Visualization of Statistics for 1D Data

We were first asked to load the P1Q1 data into Paraview and create a histogram view with a bin count of 100. I was never able to get all the bins to come together, this was the nicest that I could do.

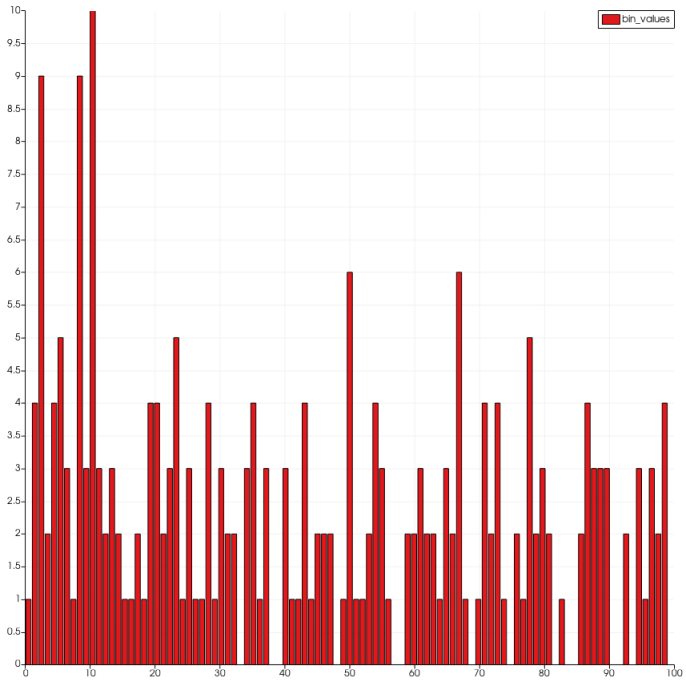

Figure 1: Histogram of the P1Q1 Data in Paraview

The number that occurred the most frequently is 10 and it occurred 10 times. There was 14 numbers that were never used by the class.

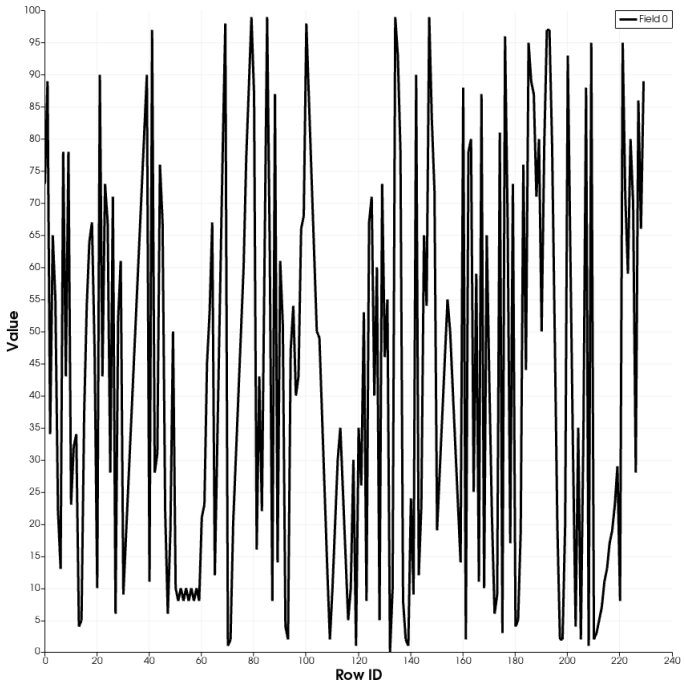

Figure 2: Line Chart of the P1Q1 Data in Paraview

We were then asked to make a line chart using the same data set as before.

Q2: Visualization of 2D Image

For question two we were asked to input the 2d.vti data into Paraview and make a histogram to aid us in finding the threshold values to visualize the river bed of the Grand Canyon. The threshold values I used for capturing the riverbed were a minimum value of 0 and a maximum value of 90. Experimenting with other values brings more white and red values into the image. I may have missed some of the lower valleys with this approach but as we were aiming for the river beds I think that this was better.

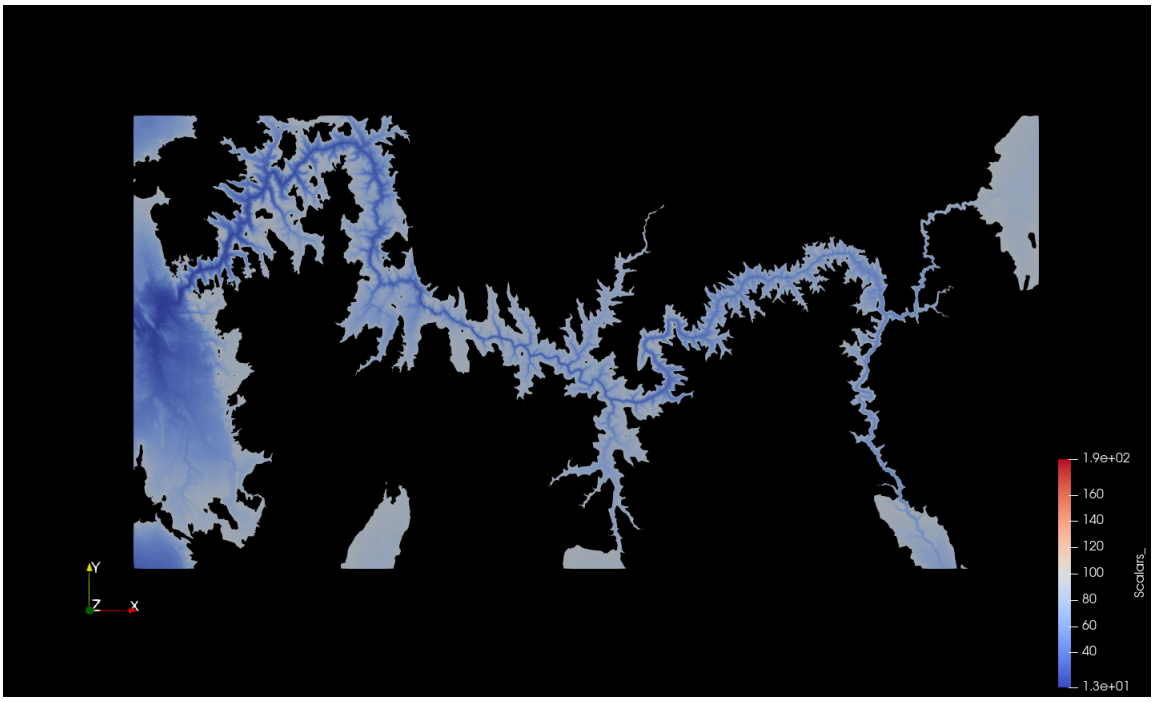

Figure 3: Threshold View of the Grand Canyon Riverbed

Looking at the information panel of Paraview the number of points in this image is 2,053,863 while the number of cells is 2,010,557. All together the format is an unstructured grid and the memory taken up is 117.79 MB.

We were also asked to create and label a contour plot of the data using multiple isocontour values.

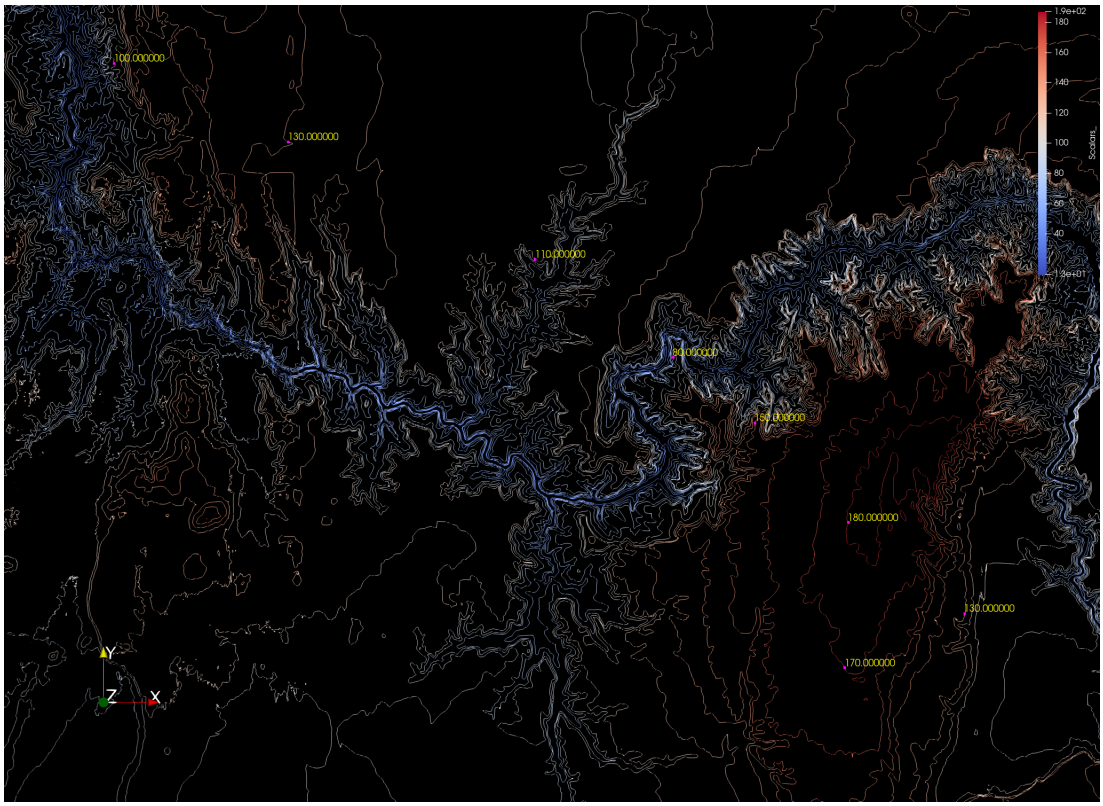

Figure 4: Contour Plot with Multiple Labeled Isocontour Values

Q3: Exploring Data on Polygonal Meshes

The minimum and maximum values that best captured the single cylinder associated with the bolts cylinder are 20 and 40 respectively. 48 venting slots were found using the slice filter and wire frame surface rendering view.

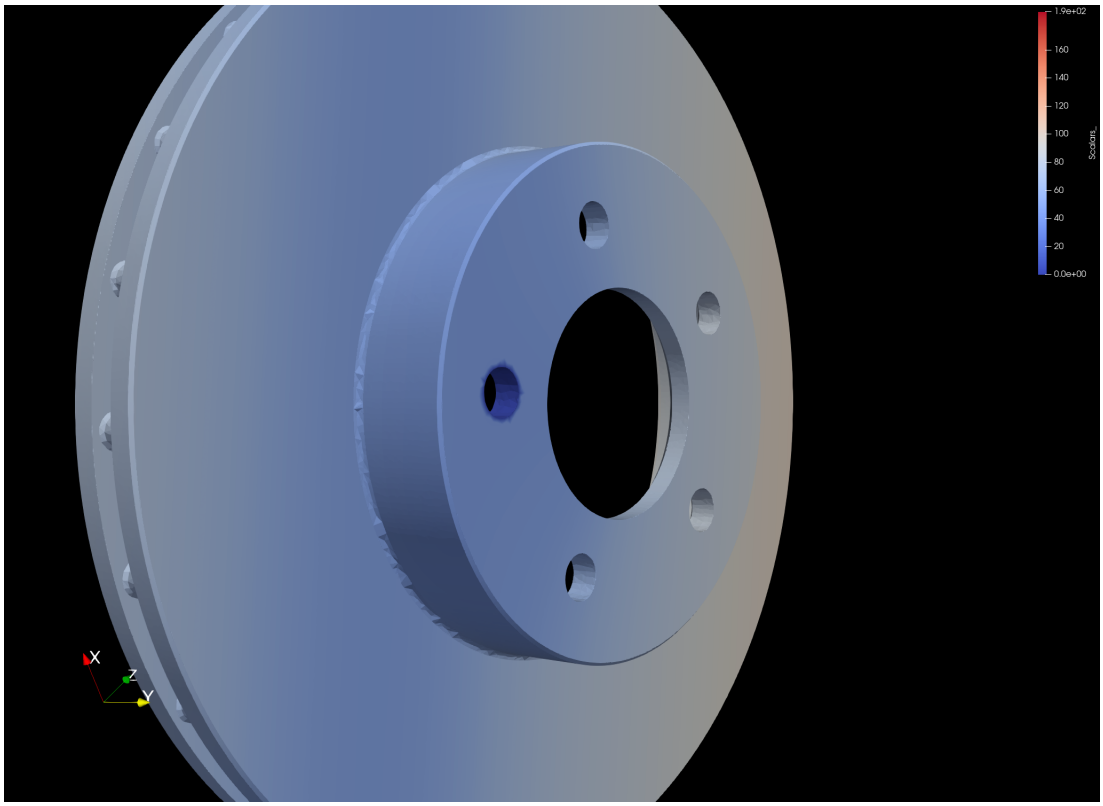

Figure 5: Bolt Cylinder

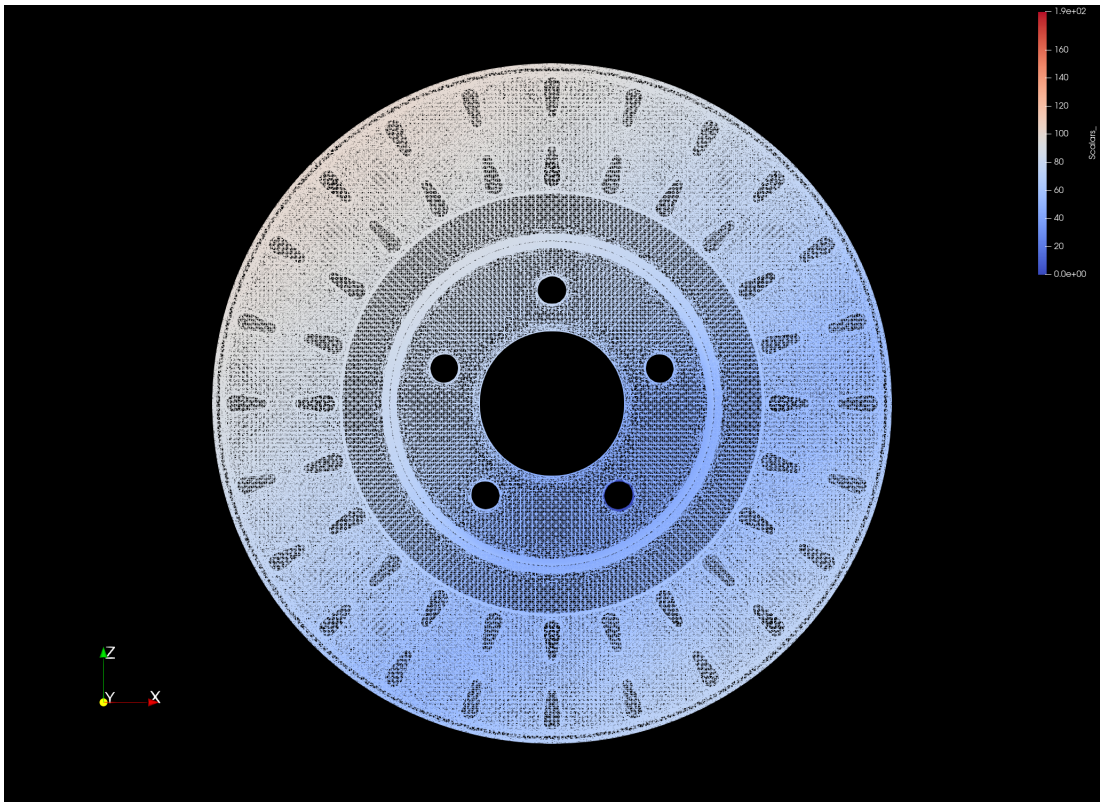

Figure 6: Ventilation Slots

Q4: Visualization of 3D Images

Next we were asked to visualize the visible woman's foot in 3D. We made three slices of the foot at the X, Y, and Z normals. Then constructed the three slices together making the unnecessary data opaque. The density of the foot can be seen to be less dense in the muscle and skin tissues and more density residing in the bones of the data.

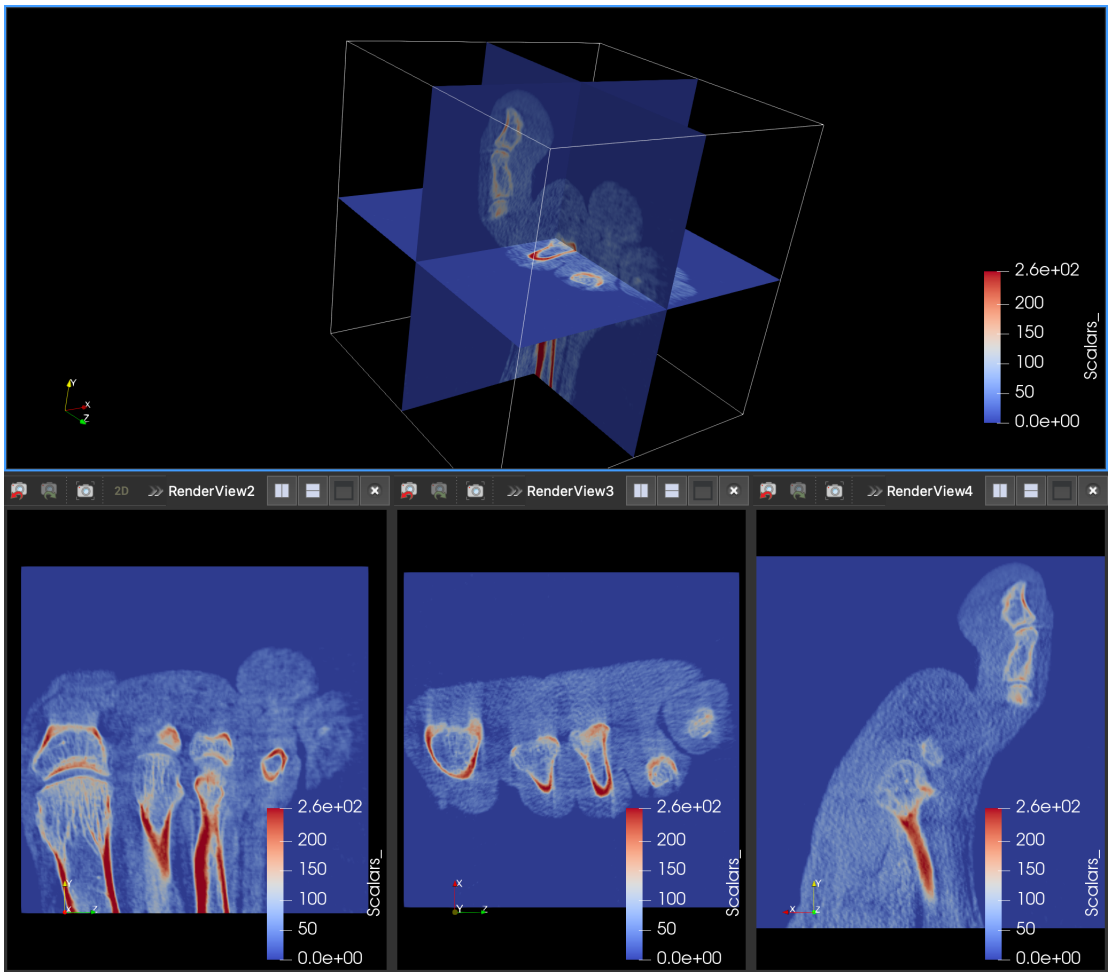

Figure 7: Generating All Slices Together for Visualization

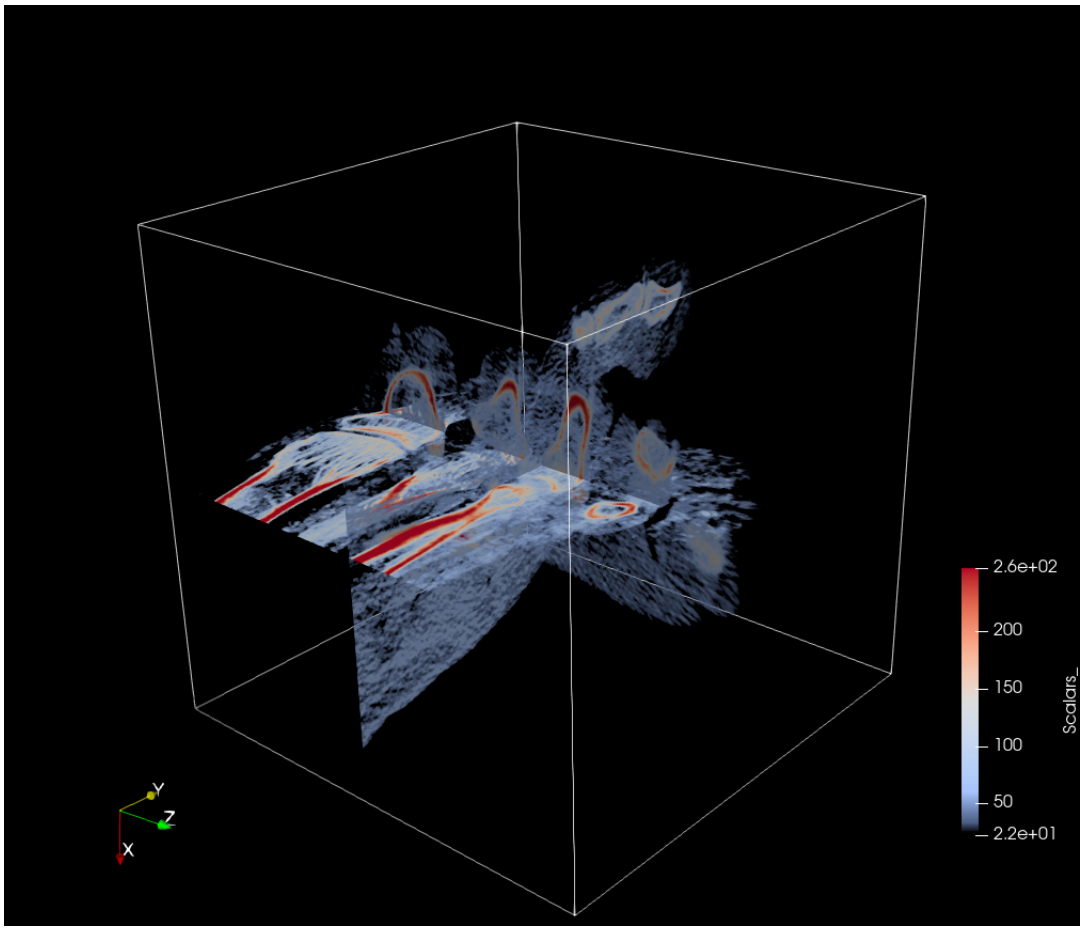

Figure 8: 3D Threshold View After Making Unnecessary Data Opaque

I also created a view of the data from a surface rendering by accident and thought that it looked cool and had a good view of the density of the model, so I am adding it here.

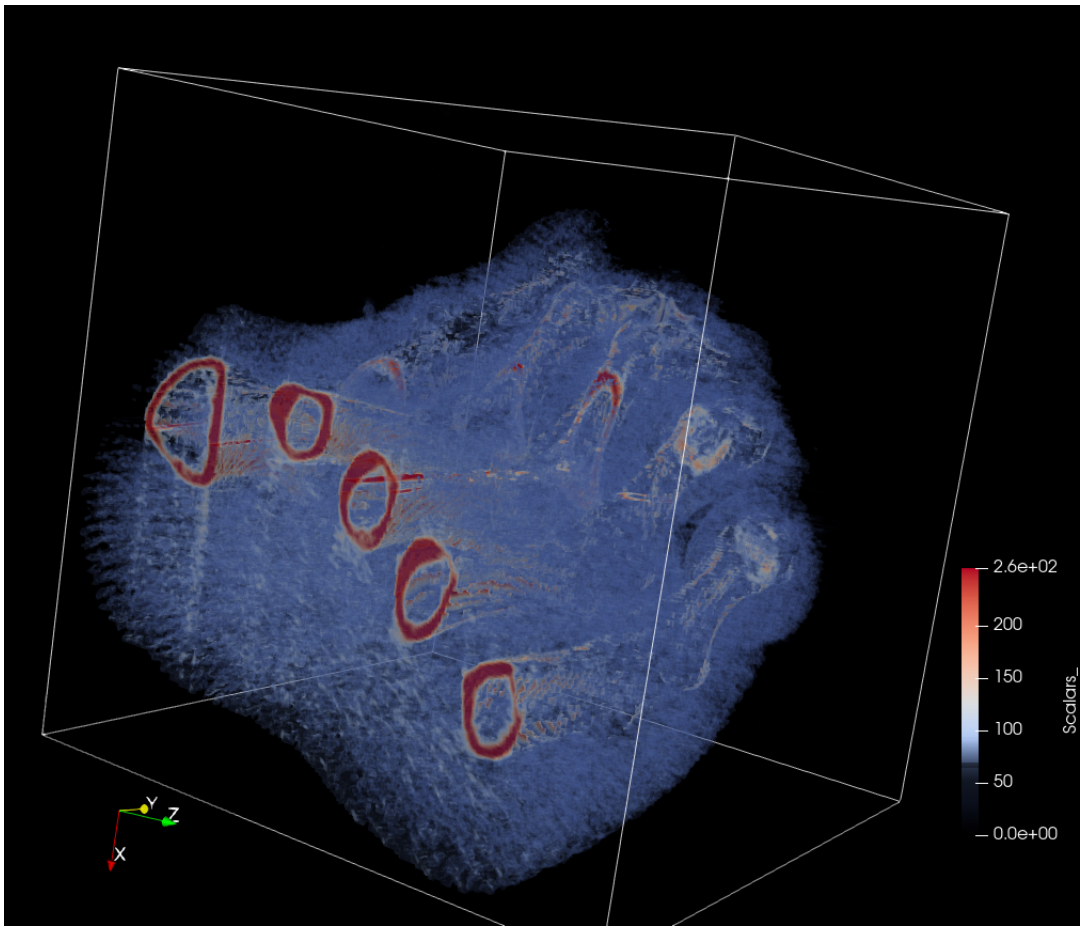

Figure 9: Cool Volume Visualization

Part 2: Code with Python Script

Q1: Use Batch Script to Create a Pipeline

For question one we were asked to create a small Python batch script to render 100 equally spaced spheres on the surface of a larger sphere. We were given a link to an Extreme Learning article [1] that talked about the math behind mapping a Fibonacci lattice and giving a little helper code. I then opened the Python script window in Paraview, started a trace, made a sphere, and then stopped the trace. This gave me the Python script of how to create a sphere in Paraview. Taking this script and putting the code to make a sphere into a method, i used the helper code from Extreme Learning to make arrays of points for the 100 equally spaced locations and went through the arrays using a for loop and plugging the coordinates into the method for making a sphere. This allowed for many spheres to be made with ease.

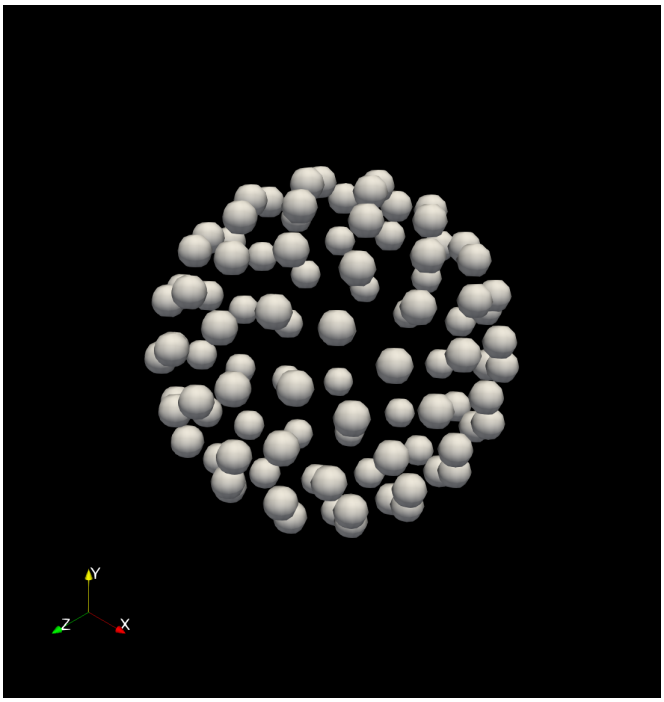

Figure 10: 100 Equally Spaced Spheres on the Surface of a Larger Sphere

Next, we were asked to create an animation using the camera to orbit around the data. Creating a perfect orbit was difficult and I was unsure of how to do this exactly, so I sided for having the orbit be a little off. The orbit takes 10 seconds with 200 frames at 20 frames per second.

We were then asked to change the radius of the 100 spheres by changing the radius of the spheres to be the absolute z value. I scaled this value using the original radius. I multiplied the original radius by the absolute value of the coordinate and got this.

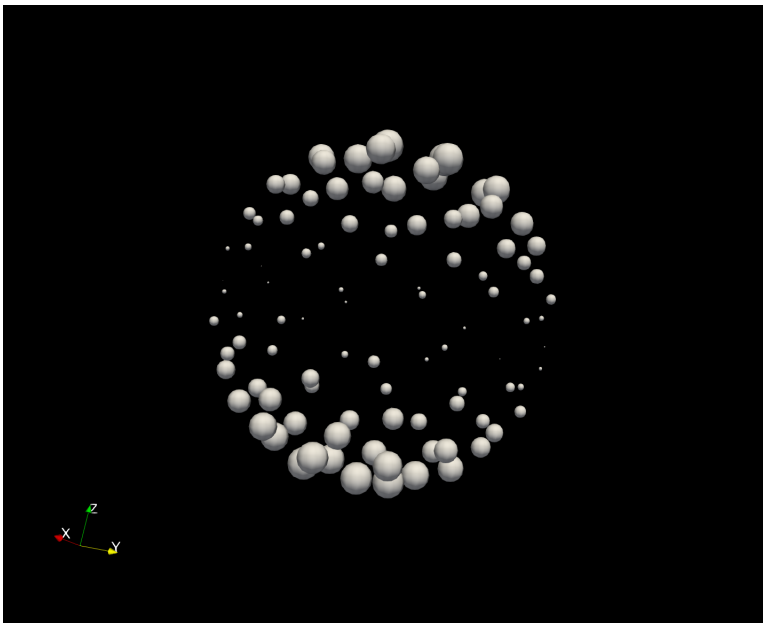

Figure 11: 100 Equally Spaced Spheres with Differing Radii According to abs(z)

Lastly, for the second part of this question we were asked to generate a Python script file that loaded the "2d.vti" data, plotted the scalar data using the PlotOverLine filter and then asked to generate a 3D elevation map of the data. For this, I started by creating a trace for the Python script. Then, I used the "Extract Surface" filter to get the data of the elevation. Then using the "Linear Extrusion" filter the elevation data was set in a form that when the plugin "Extrusion Surface" was used the desired elevation map was created. Lastly, I stopped the trace and received the desired Python script and saved it. This is what the data looked like.

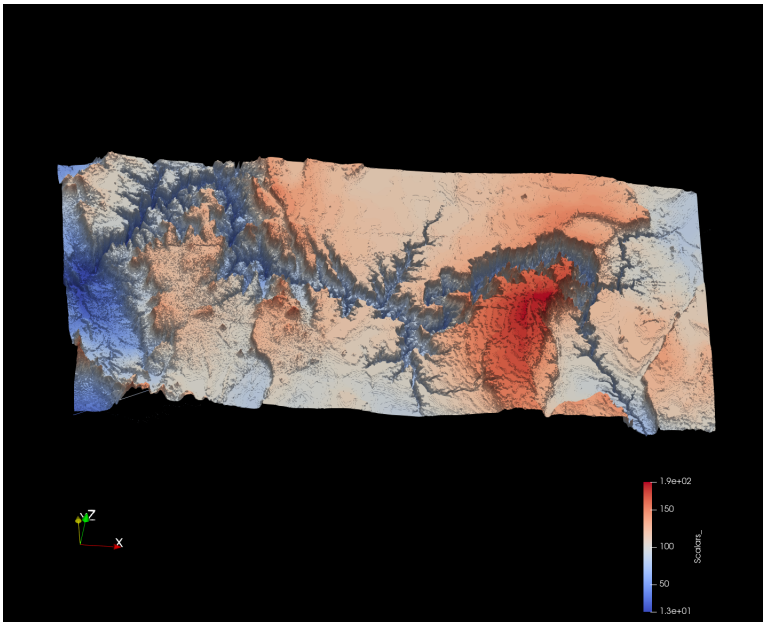

Figure 12: Extrusion Surface of the Elevation Data

The visualization shows that in the river basin area the elevation is much lower, shown by the dark blue area. The white and lighter red areas are higher with the dark red area in the lower right being having the greatest elevation in the visualization.

Conclusion

There are many ways of visualizing data in Paraview that give you a better understanding of the data you are working with. There are also many tools to better work with the visualizations that Paraview has. Python scripting allows for streamlining of processes so that, in our case, the user does not have to generate 100 different spheres at 100 different locations manually but can instead generate a method for making spheres with specified coordinates, gereate arrays containing information about the coordinates, and then plug them all in in the span of a couple minutes saving hours of time. Contour plots and 3D elevation maps can be used to see what the area would look like in real life as well as label elevation values. This lets the user quickly see desired areas in the data and draw conclusions faster. Lastly, 3D images can be made for visualization of medical data such as the foot in part 1 question 4 where more intuition about the data is made from seeing the foot in 3D. If the user is looking for dense parts of the foot and wants to see if there is a fracture or break of some sorts, they only have to highlight the data as seen in order to draw conclusions from the data.

References

[1] "Extreme Learning" : http://extremelearning.com.au/evenly-distributing-points-on-a-sphere/

Appendix

Python script for evenly spaced spheres.

```
#  trace generated using paraview version 5.11.0
# import paraview
#paraview.compatibility.major = 5
#paraview.compatibility.minor = 11

#### import the simple module from the paraview
from paraview.simple import *
import numpy as np

#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

def SphereMaker ( _sphereName, _radius, _centerX, _centerY, _centerZ ):

    # create a new 'Sphere'
    sphere1 = Sphere( registrationName = _sphereName, Radius = _radius, Center = [
                                                    _centerX, _centerY, _centerZ ] )

    # set active source
    SetActiveSource(sphere1)

    # get active view
    renderView1 = GetActiveViewOrCreate('RenderView')

    # show data in view
    sphere1Display = Show(sphere1, renderView1, 'GeometryRepresentation')

    # trace defaults for the display properties.
    sphere1Display.Representation = 'Surface'
    sphere1Display.ColorArrayName = [None, '']
    sphere1Display.SelectTCoordArray = 'None'
    sphere1Display.SelectNormalArray = 'Normals'
    sphere1Display.SelectTangentArray = 'None'
    sphere1Display.OSPRayScaleArray = 'Normals'
    sphere1Display.OSPRayScaleFunction = 'PiecewiseFunction'
    sphere1Display.SelectOrientationVectors = 'None'
    sphere1Display.ScaleFactor = 0.1
    sphere1Display.SelectScaleArray = 'None'
    sphere1Display.GlyphType = 'Arrow'
    sphere1Display.GlyphTableIndexArray = 'None'
    sphere1Display.GaussianRadius = 0.005
    sphere1Display.SetScaleArray = ['POINTS', 'Normals']
    sphere1Display.ScaleTransferFunction = 'PiecewiseFunction'
    sphere1Display.OpacityArray = ['POINTS', 'Normals']
    sphere1Display.OpacityTransferFunction = 'PiecewiseFunction'
    sphere1Display.DataAxesGrid = 'GridAxesRepresentation'
    sphere1Display.PolarAxes = 'PolarAxesRepresentation'
    sphere1Display.SelectInputVectors = ['POINTS', 'Normals']
    sphere1Display.WriteLog = ''

    # init the 'PiecewiseFunction' selected for 'ScaleTransferFunction'
    sphere1Display.ScaleTransferFunction.Points = [-0.9749279022216797, 0.0, 0.5, 0.0, 0
                                                   .9749279022216797, 1.0, 0.5, 0.0]

    # init the 'PiecewiseFunction' selected for 'OpacityTransferFunction'
    sphere1Display.OpacityTransferFunction.Points = [-0.9749279022216797, 0.0, 0.5, 0.0,
                                                     0.9749279022216797, 1.0, 0.5, 0.0]

    # get the material library
    materialLibrary1 = GetMaterialLibrary()

    # reset view to fit data
    renderView1.ResetCamera(False)

    # update the view to ensure updated data information
    renderView1.Update()
```

```
    #================================================================
    # addendum: following script captures some of the application
    # state to faithfully reproduce the visualization during playback
    #================================================================

    # get layout
    layout1 = GetLayout()

    #--------------------------------
    # saving layout sizes for layouts

    # layout/tab size in pixels
    layout1.SetSize(956, 970)

    #-----------------------------------
    # saving camera placements for views

    # current camera placement for renderView1
    renderView1.CameraPosition = [2.8232517173479246, -0.699530704008952, -1.
                                                    6323267077449688]
    renderView1.CameraViewUp = [0.12212900873005147, -0.8181689804280626, 0.
                                                    5618576551867202]
    renderView1.CameraParallelScale = 0.8640828340587008

    #--------------------------------------------
    # uncomment the following to render all views
    # RenderAllViews()
    # alternatively, if you want to write images, you can use SaveScreenshot(...).


# Run my code here

numberOfSpheres = 100
distanceSpacing = 0.5
distancingArray = np.arange( 0, numberOfSpheres, dtype = float ) + distanceSpacing
phiArray = np.arccos( 1 - 2 * distancingArray / numberOfSpheres )
goldenRatio = ( 1 + 5**0.5 ) / 2
thetaArray = 2 * np.pi * distancingArray / goldenRatio

radius = 0.1
# SphereMaker( 'Start', radius, 0, 0, 0 )

for i in range( 0, numberOfSpheres ):

    sphereName = 'sphere ' + str(i)

    # Get the center locations
    x = np.cos( thetaArray[ i ] ) * np.sin( phiArray[ i ] )
    y = np.sin( thetaArray[ i ] ) * np.sin( phiArray[ i ] )
    z = np.cos( phiArray[ i ] )

    # Contents : Name , Radius, centerX, centerY, centerZ
    SphereMaker( sphereName, radius, x, y, z )
    # SphereMaker( sphereName, abs( z ) * radius, x, y, z )

print('Code completed.')
```

Python script for extrusion surface.

```
# trace generated using paraview version 5.11.0
#import paraview
#paraview.compatibility.major = 5
#paraview.compatibility.minor = 11

#### import the simple module from the paraview
from paraview.simple import *
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# create a new 'XML Image Data Reader'
a2dvti = XMLImageDataReader(registrationName='2d.vti', FileName=['/Users/smerkley95/
                                                  Library/CloudStorage/SynologyDrive/School/
                                                  UofU/CS 5635 (2023)/Hw/Assignment02 -
                                                  Scalar Fields with Paraview/data02/2d.vti'
                                                  ])
a2dvti.PointArrayStatus = ['Scalars_']

# Properties modified on a2dvti
a2dvti.TimeArray = 'None'

# get active view
renderView1 = GetActiveViewOrCreate('RenderView')

# show data in view
a2dvtiDisplay = Show(a2dvti, renderView1, 'UniformGridRepresentation')

# get color transfer function/color map for 'Scalars_'
scalars_LUT = GetColorTransferFunction('Scalars_')

# get opacity transfer function/opacity map for 'Scalars_'
scalars_PWF = GetOpacityTransferFunction('Scalars_')

# trace defaults for the display properties.
a2dvtiDisplay.Representation = 'Slice'
a2dvtiDisplay.ColorArrayName = ['POINTS', 'Scalars_']
a2dvtiDisplay.LookupTable = scalars_LUT
a2dvtiDisplay.SelectTCoordArray = 'None'
a2dvtiDisplay.SelectNormalArray = 'None'
a2dvtiDisplay.SelectTangentArray = 'None'
a2dvtiDisplay.OSPRayScaleArray = 'Scalars_'
a2dvtiDisplay.OSPRayScaleFunction = 'PiecewiseFunction'
a2dvtiDisplay.SelectOrientationVectors = 'None'
a2dvtiDisplay.ScaleFactor = 409.6
a2dvtiDisplay.SelectScaleArray = 'Scalars_'
a2dvtiDisplay.GlyphType = 'Arrow'
a2dvtiDisplay.GlyphTableIndexArray = 'Scalars_'
a2dvtiDisplay.GaussianRadius = 20.48
a2dvtiDisplay.SetScaleArray = ['POINTS', 'Scalars_']
a2dvtiDisplay.ScaleTransferFunction = 'PiecewiseFunction'
a2dvtiDisplay.OpacityArray = ['POINTS', 'Scalars_']
a2dvtiDisplay.OpacityTransferFunction = 'PiecewiseFunction'
a2dvtiDisplay.DataAxesGrid = 'GridAxesRepresentation'
a2dvtiDisplay.PolarAxes = 'PolarAxesRepresentation'
a2dvtiDisplay.ScalarOpacityUnitDistance = 22.538152910782724
a2dvtiDisplay.ScalarOpacityFunction = scalars_PWF
a2dvtiDisplay.OpacityArrayName = ['POINTS', 'Scalars_']
a2dvtiDisplay.ColorArray2Name = ['POINTS', 'Scalars_']
a2dvtiDisplay.IsosurfaceValues = [100.0]
a2dvtiDisplay.SliceFunction = 'Plane'
a2dvtiDisplay.SelectInputVectors = [None, '']
a2dvtiDisplay.WriteLog = ''
a2dvtiDisplay.BumpMapInputDataArray = ['POINTS', 'Scalars_']
a2dvtiDisplay.ExtrusionInputDataArray = ['POINTS', 'Scalars_']

# init the 'PiecewiseFunction' selected for 'ScaleTransferFunction'
a2dvtiDisplay.ScaleTransferFunction.Points = [13.0, 0.0, 0.5, 0.0, 187.0, 1.0, 0.5, 0.0]

# init the 'PiecewiseFunction' selected for 'OpacityTransferFunction'
```

```
a2dvtiDisplay.OpacityTransferFunction.Points = [13.0, 0.0, 0.5, 0.0, 187.0, 1.0, 0.5, 0.
                                                0]

# init the 'Plane' selected for 'SliceFunction'
a2dvtiDisplay.SliceFunction.Origin = [2048.0, 1024.0, 0.0]

# reset view to fit data
renderView1.ResetCamera(False)

#changing interaction mode based on data extents
renderView1.InteractionMode = '2D'
renderView1.CameraPosition = [2048.0, 1024.0, 10000.0]
renderView1.CameraFocalPoint = [2048.0, 1024.0, 0.0]

# get the material library
materialLibrary1 = GetMaterialLibrary()

# show color bar/color legend
a2dvtiDisplay.SetScalarBarVisibility(renderView1, True)

# update the view to ensure updated data information
renderView1.Update()

# get 2D transfer function for 'Scalars_'
scalars_TF2D = GetTransferFunction2D('Scalars_')

# create a new 'Plot Over Line'
plotOverLine1 = PlotOverLine(registrationName='PlotOverLine1', Input=a2dvti)
plotOverLine1.Point2 = [4096.0, 2048.0, 0.0]

# show data in view
plotOverLine1Display = Show(plotOverLine1, renderView1, 'GeometryRepresentation')

# trace defaults for the display properties.
plotOverLine1Display.Representation = 'Surface'
plotOverLine1Display.ColorArrayName = ['POINTS', 'Scalars_']
plotOverLine1Display.LookupTable = scalars_LUT
plotOverLine1Display.SelectTCoordArray = 'None'
plotOverLine1Display.SelectNormalArray = 'None'
plotOverLine1Display.SelectTangentArray = 'None'
plotOverLine1Display.OSPRayScaleArray = 'Scalars_'
plotOverLine1Display.OSPRayScaleFunction = 'PiecewiseFunction'
plotOverLine1Display.SelectOrientationVectors = 'None'
plotOverLine1Display.ScaleFactor = 409.6
plotOverLine1Display.SelectScaleArray = 'Scalars_'
plotOverLine1Display.GlyphType = 'Arrow'
plotOverLine1Display.GlyphTableIndexArray = 'Scalars_'
plotOverLine1Display.GaussianRadius = 20.48
plotOverLine1Display.SetScaleArray = ['POINTS', 'Scalars_']
plotOverLine1Display.ScaleTransferFunction = 'PiecewiseFunction'
plotOverLine1Display.OpacityArray = ['POINTS', 'Scalars_']
plotOverLine1Display.OpacityTransferFunction = 'PiecewiseFunction'
plotOverLine1Display.DataAxesGrid = 'GridAxesRepresentation'
plotOverLine1Display.PolarAxes = 'PolarAxesRepresentation'
plotOverLine1Display.SelectInputVectors = [None, '']
plotOverLine1Display.WriteLog = ''
plotOverLine1Display.BumpMapInputDataArray = ['POINTS', 'Scalars_']
plotOverLine1Display.ExtrusionInputDataArray = ['POINTS', 'Scalars_']

# init the 'PiecewiseFunction' selected for 'ScaleTransferFunction'
plotOverLine1Display.ScaleTransferFunction.Points = [29.0, 0.0, 0.5, 0.0, 150.0, 1.0, 0.
                                                5, 0.0]

# init the 'PiecewiseFunction' selected for 'OpacityTransferFunction'
plotOverLine1Display.OpacityTransferFunction.Points = [29.0, 0.0, 0.5, 0.0, 150.0, 1.0,
                                                0.5, 0.0]

# Create a new 'Line Chart View'
lineChartView1 = CreateView('XYChartView')

# show data in view
plotOverLine1Display_1 = Show(plotOverLine1, lineChartView1, 'XYChartRepresentation')
```

```
# trace defaults for the display properties.
plotOverLine1Display_1.UseIndexForXAxis = 0
plotOverLine1Display_1.XArrayName = 'arc_length'
plotOverLine1Display_1.SeriesVisibility = ['Scalars_']
plotOverLine1Display_1.SeriesLabel = ['arc_length', 'arc_length', 'Scalars_', 'Scalars_'
                                      , 'vtkValidPointMask', 'vtkValidPointMask'
                                      , 'Points_X', 'Points_X', 'Points_Y', '
                                      Points_Y', 'Points_Z', 'Points_Z', '
                                      Points_Magnitude', 'Points_Magnitude']
plotOverLine1Display_1.SeriesColor = ['arc_length', '0', '0', '0', 'Scalars_', '0.
                                      8899977111467154', '0.10000762951094835',
                                      '0.1100022888532845', 'vtkValidPointMask',
                                       '0.220004577706569', '0.4899977111467155'
                                      , '0.7199969481956207', 'Points_X', '0.
                                      30000762951094834', '0.6899977111467155',
                                      '0.2899977111467155', 'Points_Y', '0.6', '
                                      0.3100022888532845', '0.6399938963912413',
                                       'Points_Z', '1', '0.5000076295109483', '0
                                      ', 'Points_Magnitude', '0.6500038147554742
                                      ', '0.3400015259021897', '0.
                                      16000610360875867']
plotOverLine1Display_1.SeriesOpacity = ['arc_length', '1.0', 'Scalars_', '1.0', '
                                      vtkValidPointMask', '1.0', 'Points_X', '1.
                                      0', 'Points_Y', '1.0', 'Points_Z', '1.0',
                                      'Points_Magnitude', '1.0']
plotOverLine1Display_1.SeriesPlotCorner = ['arc_length', '0', 'Scalars_', '0', '
                                      vtkValidPointMask', '0', 'Points_X', '0',
                                      'Points_Y', '0', 'Points_Z', '0', '
                                      Points_Magnitude', '0']
plotOverLine1Display_1.SeriesLabelPrefix = ''
plotOverLine1Display_1.SeriesLineStyle = ['arc_length', '1', 'Scalars_', '1', '
                                      vtkValidPointMask', '1', 'Points_X', '1',
                                      'Points_Y', '1', 'Points_Z', '1', '
                                      Points_Magnitude', '1']
plotOverLine1Display_1.SeriesLineThickness = ['arc_length', '2', 'Scalars_', '2', '
                                      vtkValidPointMask', '2', 'Points_X', '2',
                                      'Points_Y', '2', 'Points_Z', '2', '
                                      Points_Magnitude', '2']
plotOverLine1Display_1.SeriesMarkerStyle = ['arc_length', '0', 'Scalars_', '0', '
                                      vtkValidPointMask', '0', 'Points_X', '0',
                                      'Points_Y', '0', 'Points_Z', '0', '
                                      Points_Magnitude', '0']
plotOverLine1Display_1.SeriesMarkerSize = ['arc_length', '4', 'Scalars_', '4', '
                                      vtkValidPointMask', '4', 'Points_X', '4',
                                      'Points_Y', '4', 'Points_Z', '4', '
                                      Points_Magnitude', '4']

# get layout
layout1 = GetLayoutByName("Layout #1")

# add view to a layout so it's visible in UI
AssignViewToLayout(view=lineChartView1, layout=layout1, hint=0)

# Properties modified on plotOverLine1Display_1
plotOverLine1Display_1.SeriesOpacity = ['arc_length', '1', 'Scalars_', '1', '
                                      vtkValidPointMask', '1', 'Points_X', '1',
                                      'Points_Y', '1', 'Points_Z', '1', '
                                      Points_Magnitude', '1']
plotOverLine1Display_1.SeriesPlotCorner = ['Points_Magnitude', '0', 'Points_X', '0', '
                                      Points_Y', '0', 'Points_Z', '0', 'Scalars_
                                      ', '0', 'arc_length', '0', '
                                      vtkValidPointMask', '0']
plotOverLine1Display_1.SeriesLineStyle = ['Points_Magnitude', '1', 'Points_X', '1', '
                                      Points_Y', '1', 'Points_Z', '1', 'Scalars_
                                      ', '1', 'arc_length', '1', '
                                      vtkValidPointMask', '1']
plotOverLine1Display_1.SeriesLineThickness = ['Points_Magnitude', '2', 'Points_X', '2',
                                      'Points_Y', '2', 'Points_Z', '2', '
                                      Scalars_', '2', 'arc_length', '2', '
                                      vtkValidPointMask', '2']
plotOverLine1Display_1.SeriesMarkerStyle = ['Points_Magnitude', '0', 'Points_X', '0', '
```

```
                                                Points_Y', '0', 'Points_Z', '0', 'Scalars_
                                                ', '0', 'arc_length', '0', '
                                                vtkValidPointMask', '0']
plotOverLine1Display_1.SeriesMarkerSize = ['Points_Magnitude', '4', 'Points_X', '4', '
                                                Points_Y', '4', 'Points_Z', '4', 'Scalars_
                                                ', '4', 'arc_length', '4', '
                                                vtkValidPointMask', '4']

# create a new 'Extract Surface'
extractSurface1 = ExtractSurface(registrationName='ExtractSurface1', Input=plotOverLine1
                                                )

# create a new 'Linear Extrusion'
linearExtrusion1 = LinearExtrusion(registrationName='LinearExtrusion1', Input=
                                                extractSurface1)

# set active source
SetActiveSource(plotOverLine1)

# set active source
SetActiveSource(a2dvti)

# set active source
SetActiveSource(linearExtrusion1)

# set active view
SetActiveView(renderView1)

# set active source
SetActiveSource(a2dvti)

# toggle interactive widget visibility (only when running from the GUI)
ShowInteractiveWidgets(proxy=a2dvtiDisplay.SliceFunction)

# toggle interactive widget visibility (only when running from the GUI)
ShowInteractiveWidgets(proxy=a2dvtiDisplay)

# toggle interactive widget visibility (only when running from the GUI)
HideInteractiveWidgets(proxy=a2dvtiDisplay.SliceFunction)

# toggle interactive widget visibility (only when running from the GUI)
HideInteractiveWidgets(proxy=a2dvtiDisplay)

# update the view to ensure updated data information
lineChartView1.Update()

# change representation type
a2dvtiDisplay.SetRepresentationType('Extrusion Surface')

# set active source
SetActiveSource(linearExtrusion1)

# show data in view
linearExtrusion1Display = Show(linearExtrusion1, renderView1, 'GeometryRepresentation')

# trace defaults for the display properties.
linearExtrusion1Display.Representation = 'Surface'
linearExtrusion1Display.ColorArrayName = ['POINTS', 'Scalars_']
linearExtrusion1Display.LookupTable = scalars_LUT
linearExtrusion1Display.SelectTCoordArray = 'None'
linearExtrusion1Display.SelectNormalArray = 'None'
linearExtrusion1Display.SelectTangentArray = 'None'
linearExtrusion1Display.OSPRayScaleArray = 'Scalars_'
linearExtrusion1Display.OSPRayScaleFunction = 'PiecewiseFunction'
linearExtrusion1Display.SelectOrientationVectors = 'None'
linearExtrusion1Display.ScaleFactor = 409.6
linearExtrusion1Display.SelectScaleArray = 'Scalars_'
linearExtrusion1Display.GlyphType = 'Arrow'
linearExtrusion1Display.GlyphTableIndexArray = 'Scalars_'
linearExtrusion1Display.GaussianRadius = 20.48
linearExtrusion1Display.SetScaleArray = ['POINTS', 'Scalars_']
linearExtrusion1Display.ScaleTransferFunction = 'PiecewiseFunction'
```

```
linearExtrusion1Display.OpacityArray = ['POINTS', 'Scalars_']
linearExtrusion1Display.OpacityTransferFunction = 'PiecewiseFunction'
linearExtrusion1Display.DataAxesGrid = 'GridAxesRepresentation'
linearExtrusion1Display.PolarAxes = 'PolarAxesRepresentation'
linearExtrusion1Display.SelectInputVectors = [None, '']
linearExtrusion1Display.WriteLog = ''
linearExtrusion1Display.BumpMapInputDataArray = ['POINTS', 'Scalars_']
linearExtrusion1Display.ExtrusionInputDataArray = ['POINTS', 'Scalars_']

# init the 'PiecewiseFunction' selected for 'ScaleTransferFunction'
linearExtrusion1Display.ScaleTransferFunction.Points = [29.0, 0.0, 0.5, 0.0, 150.0, 1.0,
                                                        0.5, 0.0]

# init the 'PiecewiseFunction' selected for 'OpacityTransferFunction'
linearExtrusion1Display.OpacityTransferFunction.Points = [29.0, 0.0, 0.5, 0.0, 150.0, 1.
                                                       0, 0.5, 0.0]

# show color bar/color legend
linearExtrusion1Display.SetScalarBarVisibility(renderView1, True)

# update the view to ensure updated data information
renderView1.Update()

# hide data in view
Hide(plotOverLine1, renderView1)

#================================================================
# addendum: following script captures some of the application
# state to faithfully reproduce the visualization during playback
#================================================================

#--------------------------------
# saving layout sizes for layouts

# layout/tab size in pixels
layout1.SetSize(2851, 1156)

#-----------------------------------
# saving camera placements for views

# current camera placement for renderView1
renderView1.InteractionMode = '2D'
renderView1.CameraPosition = [2048.0, 1024.0, 10000.0]
renderView1.CameraFocalPoint = [2048.0, 1024.0, 0.0]
renderView1.CameraParallelScale = 2289.7336089597848

#--------------------------------------------
# uncomment the following to render all views
# RenderAllViews()
# alternatively, if you want to write images, you can use SaveScreenshot(...).
```