Visualizing 2D Spatial Data

Height Field

First of all for converting the raw format into a format which vtk can directly handle, I read two files "assignment1.pts" and "assignment1.data" and put numbers of "assignment1.data" file in the third column of "assignment1.pts" file. So, now each row of "assignment1.pts" file represent a three dimensional data point (x,y and z). Then I added some lines to the begining of this file, lines such as "# vtk DataFile Version 3" which demonstrates the version of vtk, "Output file" line which is a title and represents the file, "ASCII" which shows the type of the file, "DATASET POLYDATA" which shows type of the dataset, and "POINTS 618 float" which shows number of points along with their type. Then I saved the file as "output.txt". The code for this part is :

```
import vtk

filename1 = "/home/mina/Desktop/data_assignment1/assignment1.data"  # z
values
filename2 = "/home/mina/Desktop/data_assignment1/assignment1.pts"

filename3 = "/home/mina/Desktop/data_assignment1/output.txt"
outputfile = open(filename3, 'w')

file = open(filename1)
zvalues = file.readlines();
file.close()

pts = vtk.vtkPoints()
pts.SetNumberOfPoints(len(zvalues))

file = open(filename2)
outputfile.write("# vtk DataFile Version 3"+"\n"+"Output file"+"\n")
outputfile.write("ASCII"+"\n"+"DATASET POLYDATA \n"+"POINTS
"+str(len(zvalues))+" float \n")
counter = 0;
for line in file:
 line = line.strip();
 a = line.split(" ")
 a = filter(None, a) #remove extra spaces
 a[2] = zvalues[counter]
 counter = counter + 1
 outputfile.write(a[0]+" "+a[1]+a[2])
 pts.InsertNextPoint(float(a[0]),float(a[1]),float(a[2]))
```

You can find this code in "conversion.py" file in here.
Now, vtk can read this file by vtkDataSetReader() function, and we will use this function for reading the file in next parts of the assignement.

produce a triangulation

1) assignment1.pts and assignment1.data dataset:

For this part, first of all I read points of these two files as a three dimensional data point and add all points to a polyData object. Then I used vtkVertexGlyphFilter() to map each data point to a vertex :

```
# Add points to polydata object
polyData = vtk.vtkPolyData()
polyData.SetPoints(pts)

gf = vtk.vtkVertexGlyphFilter()
gf.SetInputConnection(polyData.GetProducerPort())
gf.Update()
```

Then for triangulating the points, I used vtkDelaunay2D() as follows:

```
# Triangulate the point
delaunay = vtk.vtkDelaunay2D()
delaunay.SetInput(polyData)
delaunay.Update()
```

and Then I created mapper and actor over delaunay object:

```
# Create Mapper and Actor
triangulatedMapper = vtk.vtkPolyDataMapper()
triangulatedMapper.SetInputConnection(delaunay.GetOutputPort())

triangulatedActor = vtk.vtkActor()
triangulatedActor.SetMapper(triangulatedMapper)
```

I also created mapper and actor over each vertex (data point), so that data points are shown in another color and with a different size, but then I commented these lines because it was not asked in the assignment, however code of this part is as follow:
(Here the color of points is set to red, size of them is set to 3)

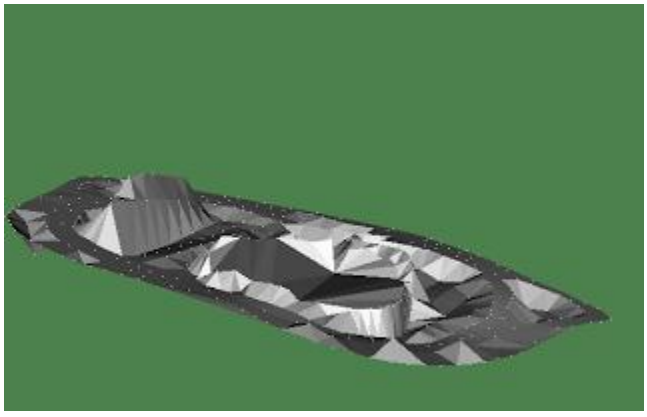
```
# Create Mapper and Actor
pointsMapper = vtk.vtkPolyDataMapper()
pointsMapper.SetInputConnection(gf.GetOutputPort())

pointsActor = vtk.vtkActor()
pointsActor.SetMapper(pointsMapper)
property = pointsActor.GetProperty()
#property.SetColor(1,0,0)
#property.SetPointSize(3)
```

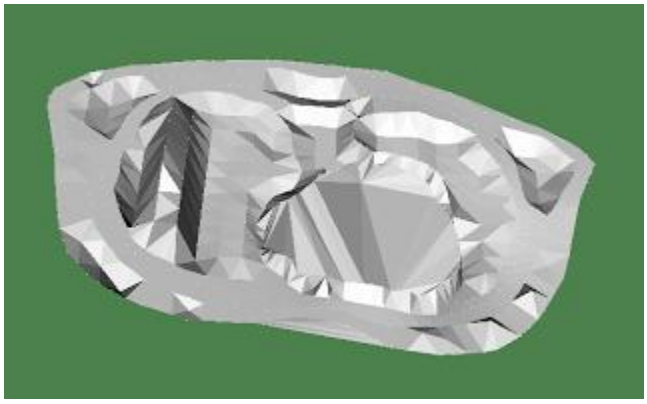
I created a renderer, a renderWindow, and a renderWindowInteractor, I added renderer to the renderWindow and set renderWindow in the renderWindowInteractor, I set the size of the window to 700*700 and its background color to green. Then at last line, I start the renderWindowInteractor:

```
# Create a renderer, render window, and interactor
renderer = vtk.vtkRenderer()
renderWindow = vtk.vtkRenderWindow()
```

```
renderWindow.SetSize(700,700)
renderWindow.AddRenderer(renderer)
renderWindowInteractor = vtk.vtkRenderWindowInteractor()
renderWindowInteractor.SetRenderWindow(renderWindow)

# Add Actor to the Scene
renderer.AddActor(pointsActor)
renderer.AddActor(triangulatedActor)
renderer.SetBackground(.3, .5, .3)

# Render and Interact
renderWindow.Render()
renderWindowInteractor.Start()
```

You can find code for this part in "Q1-1.py" file in here.
The output of this part for this data set was like the shape below, which you can play with it and turn it over itself:



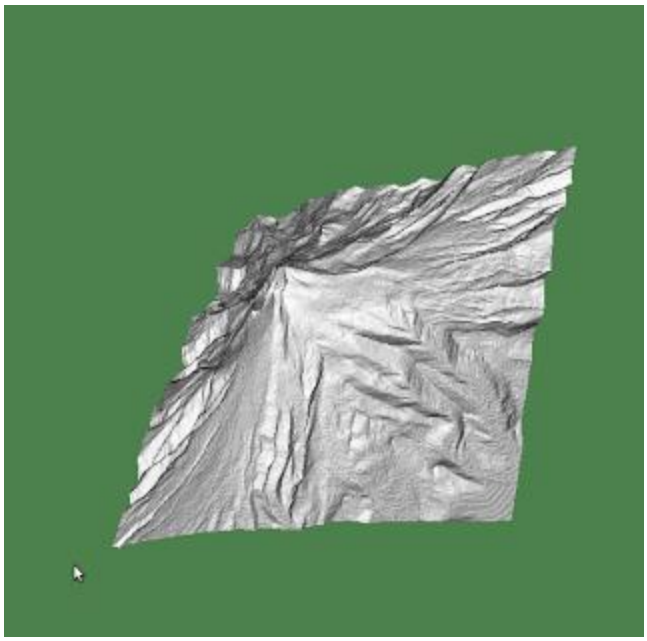


2) Mount Hood dataset:
As this data set is in .pgm format, we can read it via functions of vtkPNMReader() class. For reading this file and creating filters over it, I used vtkImageDataGeometryFilter() and vtkMergeFilter(), the code is as follow:

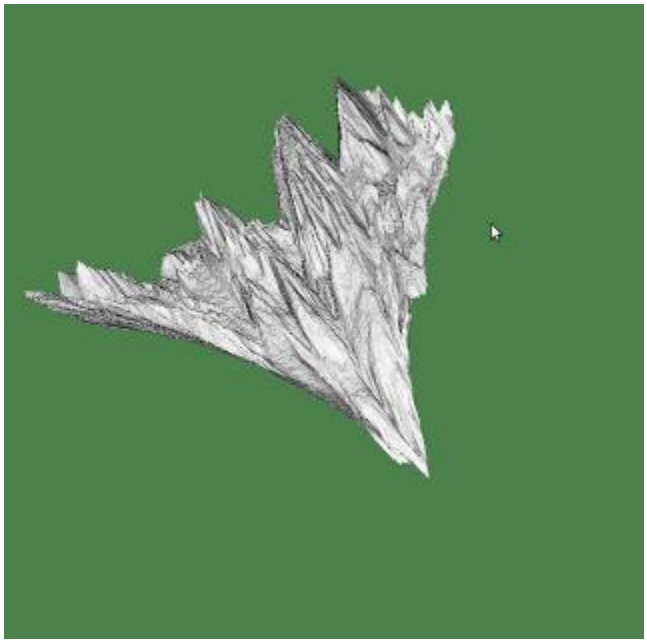
```
reader = vtk.vtkPNMReader()
reader.SetFileName(filename4)
```

```
geofilter = vtk.vtkImageDataGeometryFilter()
geofilter.SetInput(reader.GetOutput())

warp = vtk.vtkWarpScalar()
warp.SetInput(geofilter.GetOutput())
warp.SetScaleFactor(0.25)

mergefilter = vtk.vtkMergeFilter()
mergefilter.SetGeometry(warp.GetOutput())
```

The rest of the work is defining mapper and actor, and is exactly the same as previous part. you can find code for this part in "Q1-2.py" file in here. The output of this part is the mountain below which you can rotate it also.

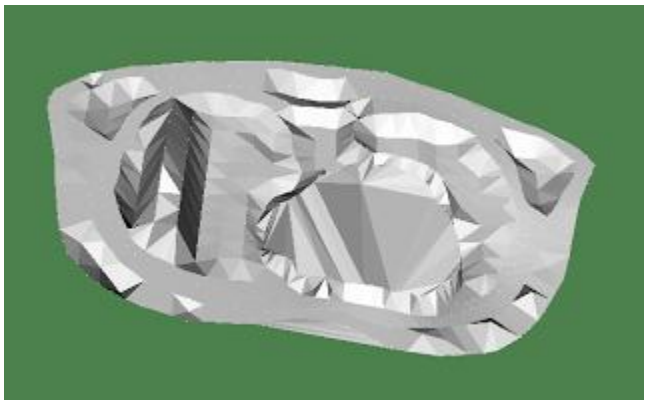


Notice that SetScaleFactor() function gets a value and scale the displacement based on that, as input value of this function is smaller, the mountain will be shown in lower elevation and wider, and as the input value is larger, the mountain will be shown in higher elevation and sharper. In the above images, ScaleFactor is set to 0.25. If we set it to 1 (which is default number) we will get images below:



As this way of displaying images, shows us images in 3D and allows us to rotate them, we can have different view of the dataset and it is a big help in understanding the dataset. On the other hand, as it uses constant coloring, it can not display low height and high height of the images

very well. for example in the image of the first dataset (shown again below) points with low height and high height seem almost the same.



I think by using coloring and assign different colors to different heights, understandability of the images will increase alot.

Contour Map

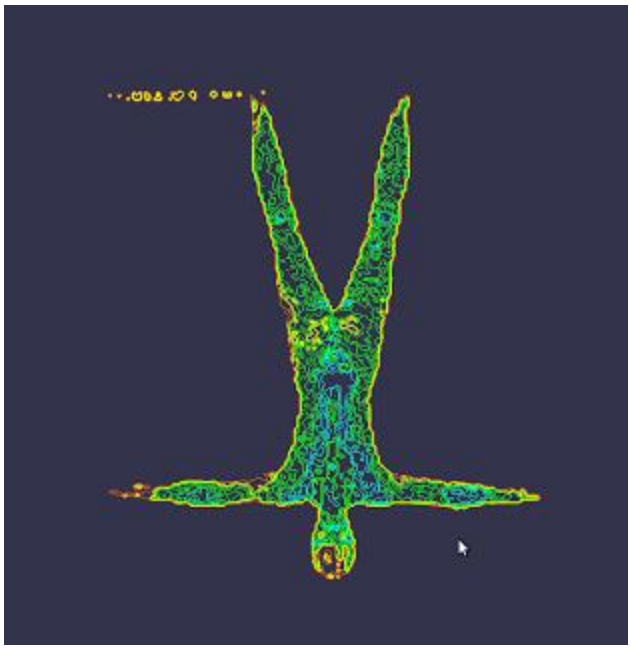
For this part, first of all I read the dataset file and then create a vtkContourFilter() :

```
reader = vtk.vtkStructuredPointsReader()
reader.SetFileName(filename1)
reader.Update()

contour = vtk.vtkContourFilter()
contour.SetInputConnection(reader.GetOutputPort())
contour.GenerateValues(10,reader.GetOutput().GetScalarRange())
```

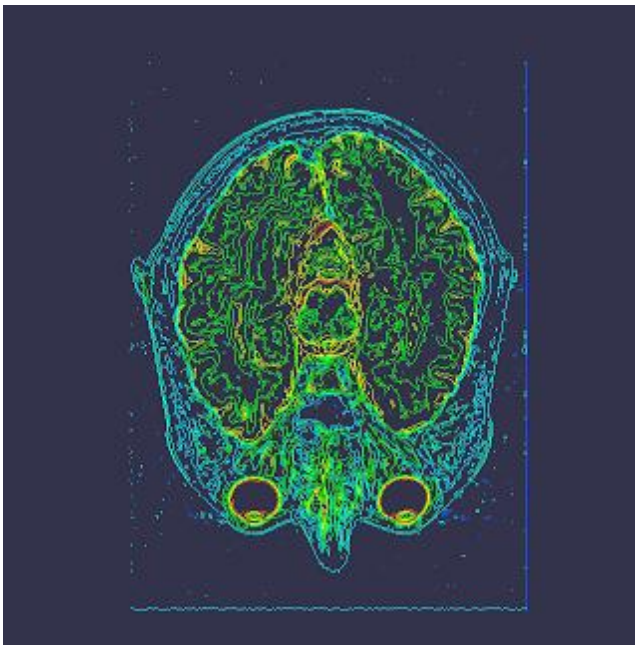
Then, I create a mapper of class vtkPolyDataMapper() and an actor:

```
contourMapper = vtk.vtkPolyDataMapper()
contourMapper.SetInputConnection(contour.GetOutputPort())
contourMapper.SetScalarRange(reader.GetOutput().GetScalarRange())

contourActor = vtk.vtkActor()
contourActor.SetMapper(contourMapper)
```

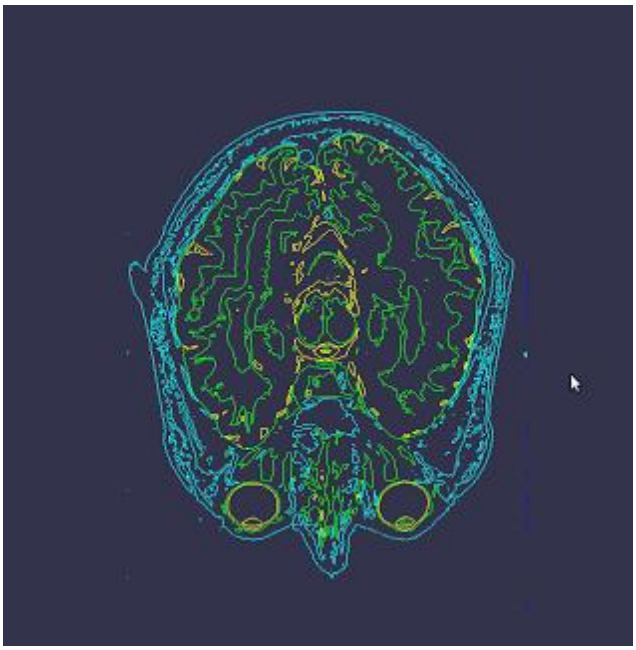
This is the image of the "body.vtk" dataset, you can find code for it in "Q2-1.py" file in here.



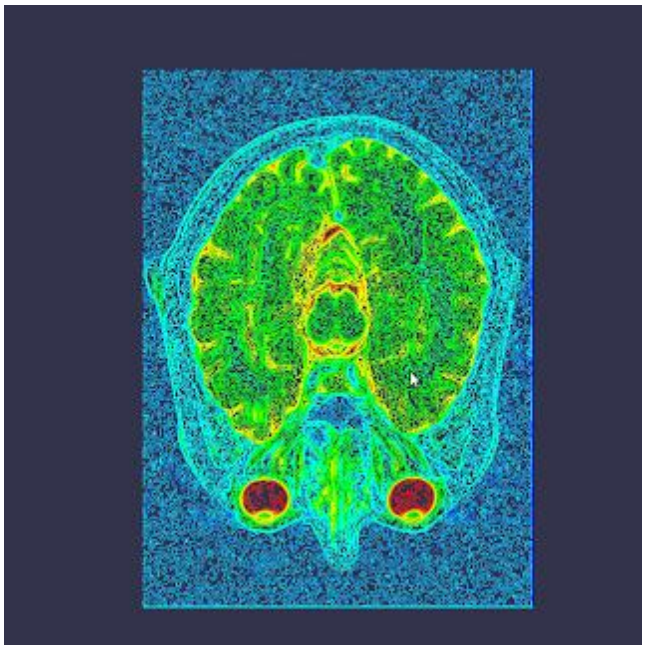
And the image below is the image of "brain.vtk" dataset, its code is totally the same as code for "body.vtk" dataset, but for ease of running the files, I created two files. code for this part is in "Q2-2.py" file.



In these two images, 10 contour values are generated and range of the values is get by reader.GetOutput().GetScalarRange(). If we increase number of contour values, we will get a different image with more details and if we decrease the number of contour values, we will get a higher level image. The image below is constructed by 5 contour values:



And the image below is constructed by 30 contour values:

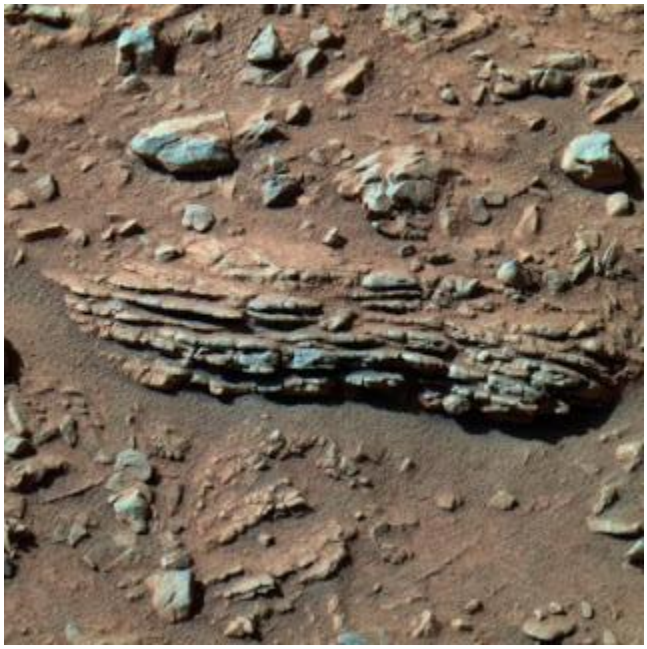


vtkContourFilter is a filter that takes as input any dataset and generates on output isosurfaces and/or isolines. The exact form of the output depends upon the dimensionality of the input data. Data consisting of 3D cells will generate isosurfaces, data consisting of 2D cells will generate isolines, and data with 1D or 0D cells will generate isopoints. By looking at the images, esp. brain image, we can see that those points around the brain are not part of the brain but are noise points, and as we compare images above, we understand that as we increase number of contour

values, more noise points are shown. So for getting rid of noise points we need to set number of contour values to a small number. On the other hand as number of contour values is smaller less details are shown in the image and image is in higher level and more abstract. So there is a trade off between displaying noise and getting a detailed image. from this point, we can say one property a dataset need to have in order to have contour lines that make sense is that, it should be noise-free or has an acceptable number of noise points. Other properties that data sets need to have (for having contour lines) include: They should have continuous data value so that contour lines can be formed at regular interval. In any given area, if the data values are increasing or decreasing, contours helps us to know how data values are changing and hence, contributes to visualization. Furthermore in any given area, a data point should not have many data points in its nearby with the same isovalue, otherwise it leads to indistinct contours.
And if I want to name a few places that these properties can be found, I can name "Elevation Maps", "Temperature Maps" and "MRI Scans" because these places have above mentioned properties.

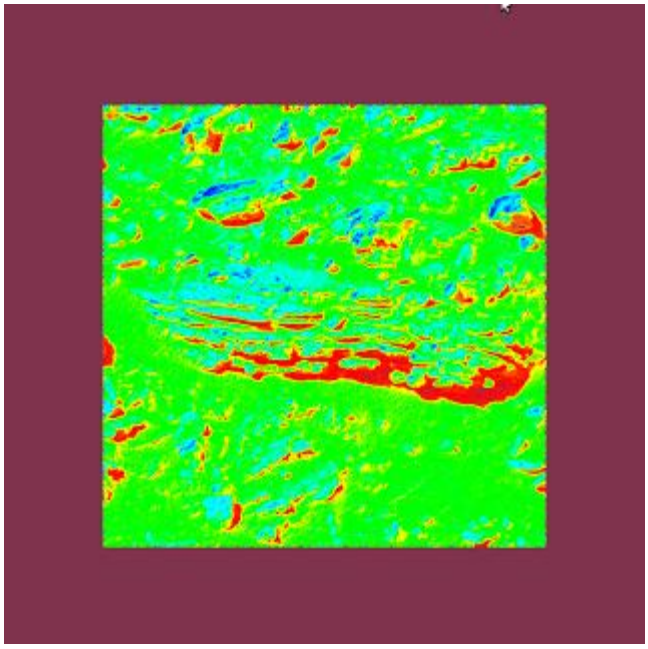
My own data set:

I chose a JPEG image of surface of Mars. The image is as follow:

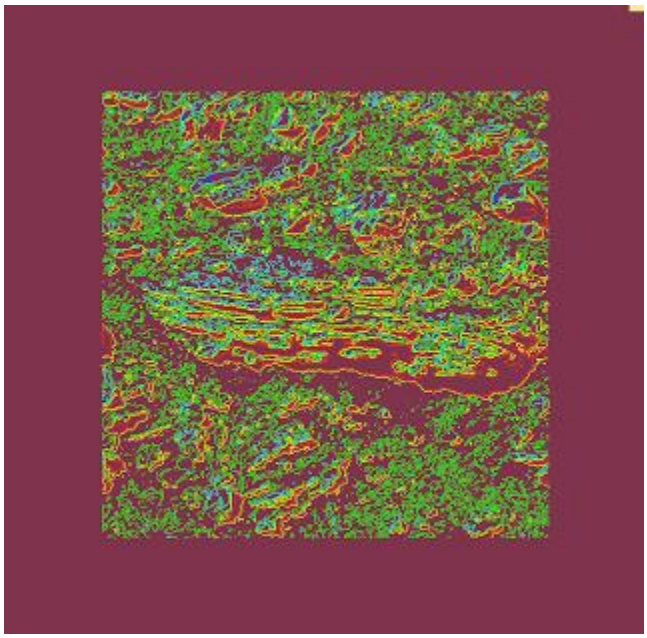


Mars surface

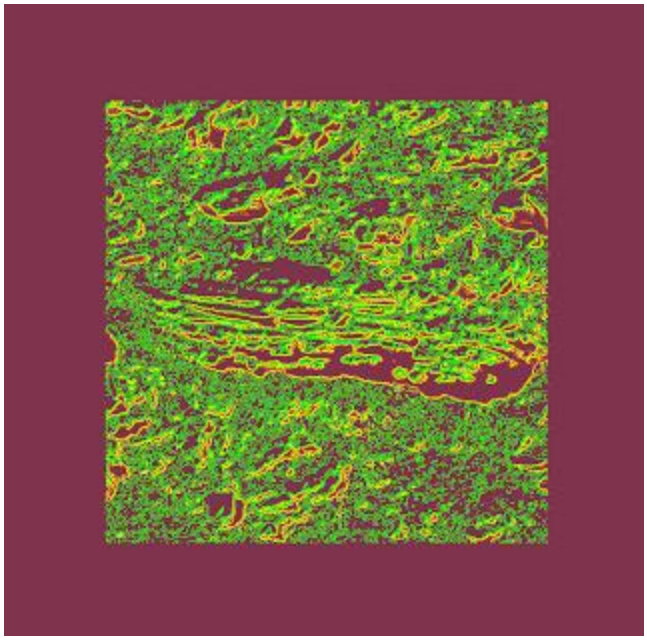
First of all, for reading this image, I used vtkJPEGReader() and then converted it to structured points dataset using vtkImageToStructuredPoints(). You can find code of this part in "Q2-3.py" file.
The images below show contour lines of it with different contour parameters.



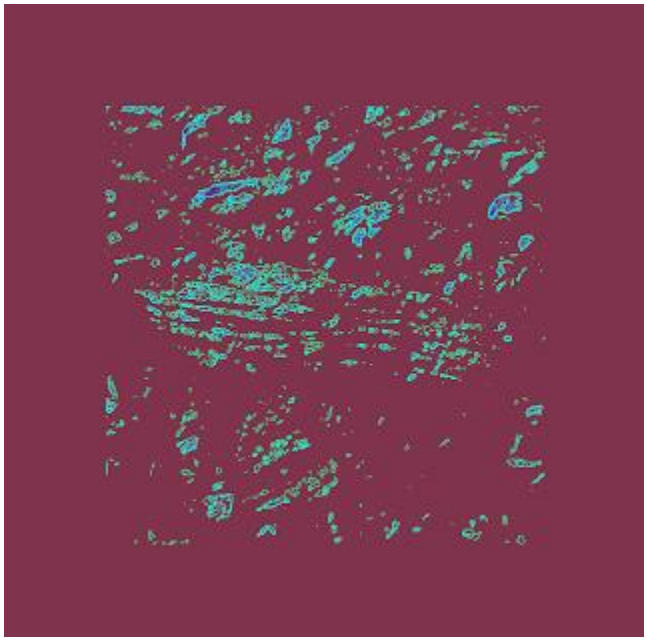
number of contours = 300



number of contours = 5, contour range = (20,225)



number of contours = 5, contour range = (30,155)



number of contours = 5, contour range = (158,255)

What data am I visualizing?
I am visualizing contours in a rock image (taken from Mars) which distinguish level of concentration of oxidized soil and dust. For example, the contour with [30-135] range shows the presence of less oxidized areas and the contour with [158-255] range shows highly oxidized areas.

I think this is interesting because:

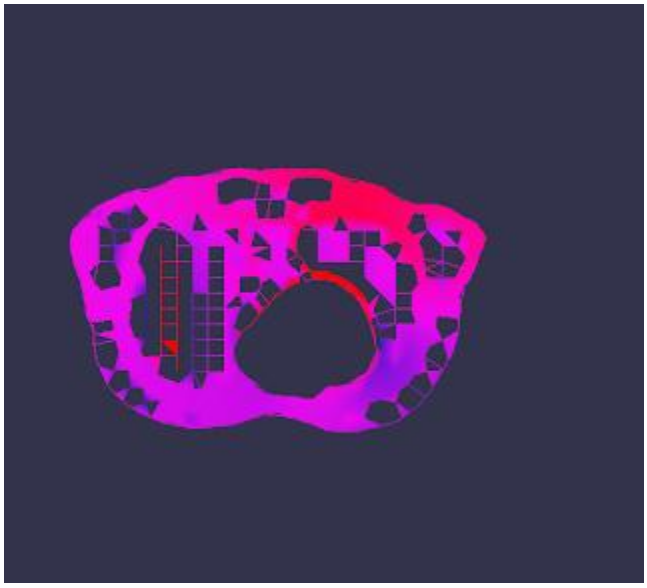
This image and corresponding contours helps us in collecting data on composition and mineralogy of rocks. This is part of one of NASA's exploration mission which aimed to figure out whether there exist any water on Mars or not.

Color Map

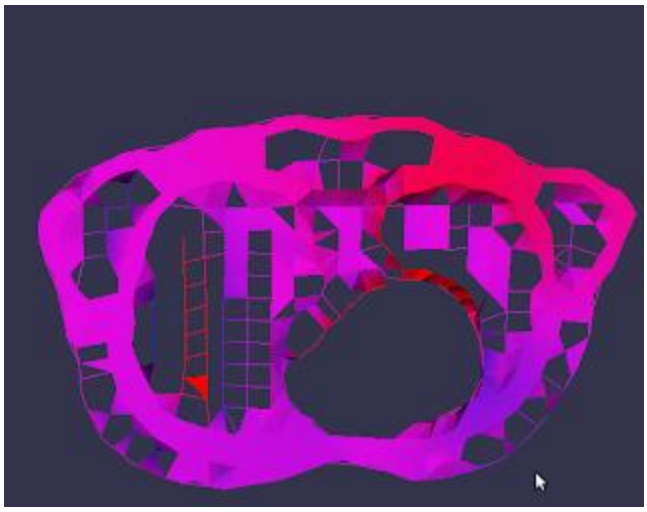
For this part, I used the same code with planer data and including vtkLookupTable() function to value colors.

For assignment1.pts and assignment1.data dataset:

- as a color-mapped plane : (you can find code for this in "Q3-1-1.py" file)

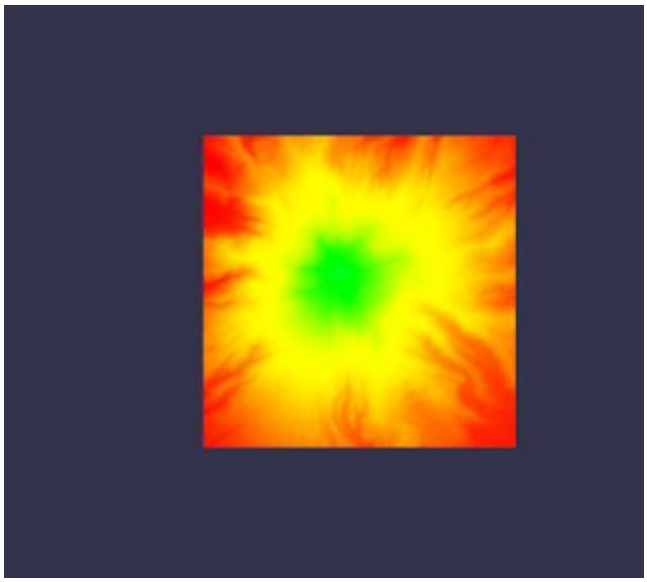


- using color map on the height field : (you can find code for this in "Q3-1-2.py" file), it is a 3D representation, you can rotate it and it gives you a better view of data compared to a color-mapped plane

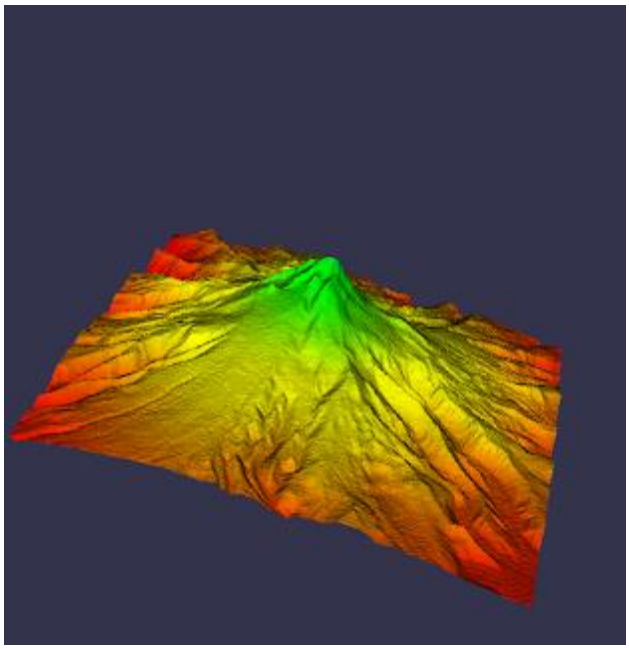


For MtHood dataset:

- as a color-mapped plane : (you can find code for this in "Q3-2-1.py" file)



- using color map on the height field : (you can find code for this in "Q3-2-2.py" file), it is a 3D representation, you can rotate it and it gives you a better view of data compared to a color-mapped plane



The benefits of the different approaches:

In the height field approach one can see elevation of data set. Contour has advantages over elevation Grid and TIN on that a group of contour lines can visually convey quantity attributes and undulation information of land surface at the same time. Readers of contour map can capture interested terrain information on global structures and local attributes of landscape efficiently. This characteristic can
be utilized to benefit digital terrain analysis, in which most functions depend on Grid
and TIN and can barely acquire information adapted to variant scales.
Contour lines are curved or straight lines on a map describing the intersection of a real or hypothetical surface with one or more horizontal planes. The configuration of these contours allows map readers to infer relative gradient of a parameter and estimate that parameter at specific places. Contour lines may be either traced on a visible three-dimensional model of the surface, as when a photogrammetrist viewing a stereo-model plots elevation contours, or interpolated from estimated surface elevations, as when a computer program threads contours through a network of observation points of area centroids.
Color map, maps data values with using color, it increase the understandability of an image, helps user get a better understanding from image.