HW02

Zhewei Wang

1 $x - y$ plot

For functional model, it descripe how the data flow in the processing steps. So in $x - y$ plot, firstly we get two sequencial data, or a function. If what we get is a function, then next step we need to geneate a discrete sequence data x to calculate y. Then next step we can use different filters for different purpose. Say if we want to show different colors on data, then we need a filter to find out the maximum and minimum value, then generate a color lookup table. Then for each point in the data, we set the RGB values to it. Then transfer the data and corresponding color value to mapper, then actor, then renderer and so on to show it on the screen.

For object model, it is kind of 'stable' and it talks about the objects in the system, and their properties and relations. So go back to our $x - y$ plot, we need reader if we want to read data from a file, or generator, if we want to generate data from a function. Then we need different filters, such as color filter which can color the data from red(min) to blue(max), size filter which can set different size to the data point, shape filter which generate different shape such as rectangle, triangle, contour filter if the function is difficult to solve. Then we need a output sink to express the data from the filter as an image. We also need a writer if we want to save the image as a file.

2 Different visualization methods

In 2D we can have properties such as color, size, position and shape. We can pick one or some from them and express the data. Let's say we want to visualize the facebook relations. So the data we have is a table, the first column is the account list, and the rest columns are the relations whith the first account. The easiest way is to set each account a point in the $x - y$, and if this point has relation with another one, then we draw a line between them. So the input is the table, output is points and 0, 1 relation between points. We can see a lot of images about this type of visualization.

The short point of this way is obviously: when the account and relations are more and more, we get nothing but chaos from the image. Another better way can be processing the data first. We can use a data cluster algorithm to separate the accounts in several large groups. Then for the same group we draw the accounts as circle and locate closely. The more relations account have, the larger the circle is. For this method, we lost the accurate connection with each other, but we can see who is the most active person in a group.

3 Visualize equations

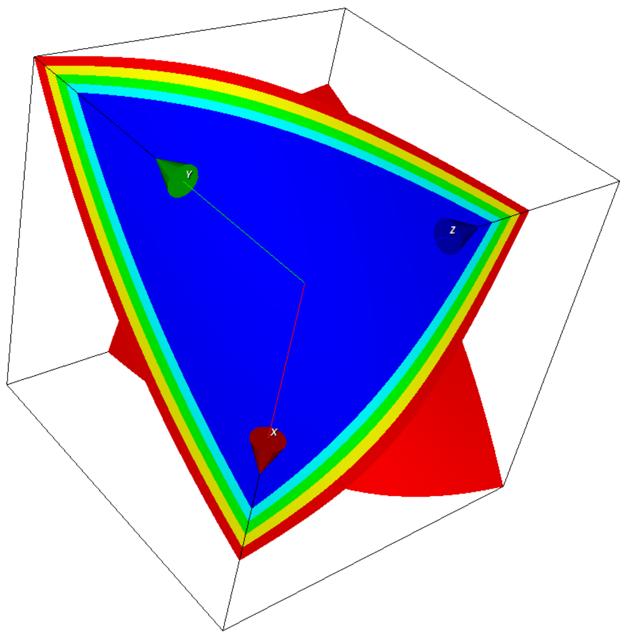

Figure 1a: Function 1

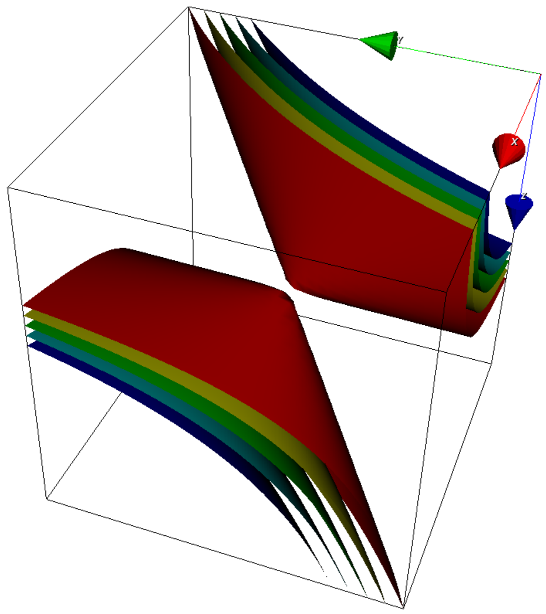

Figure 1b: Function 1

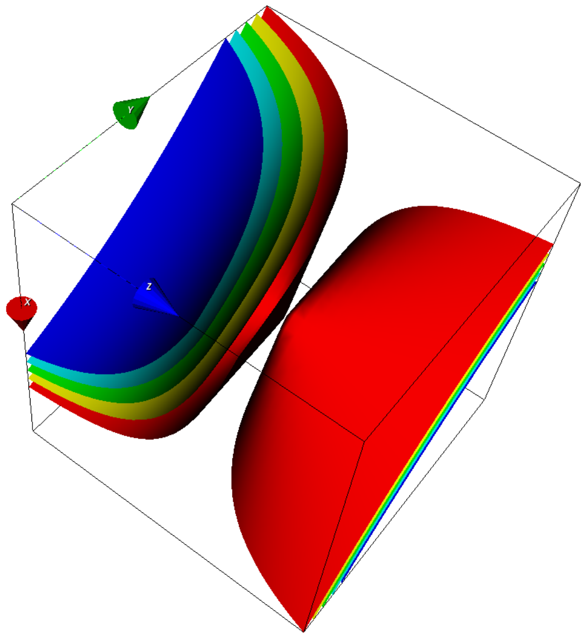

Figure 1c: Function 1

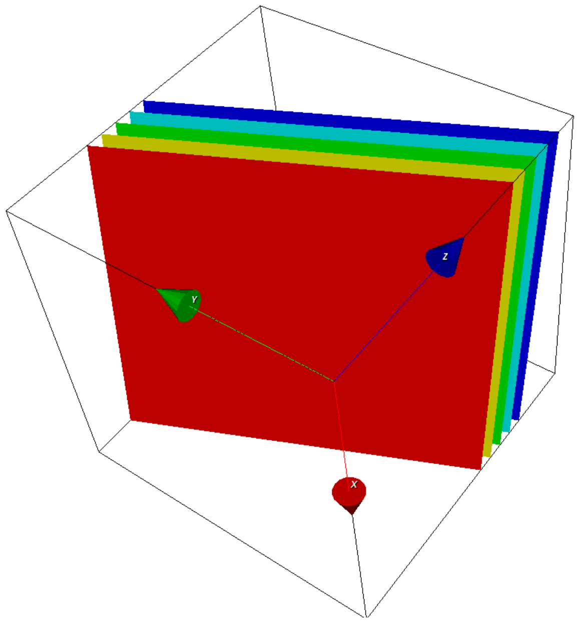

Figure 2a: Function 2

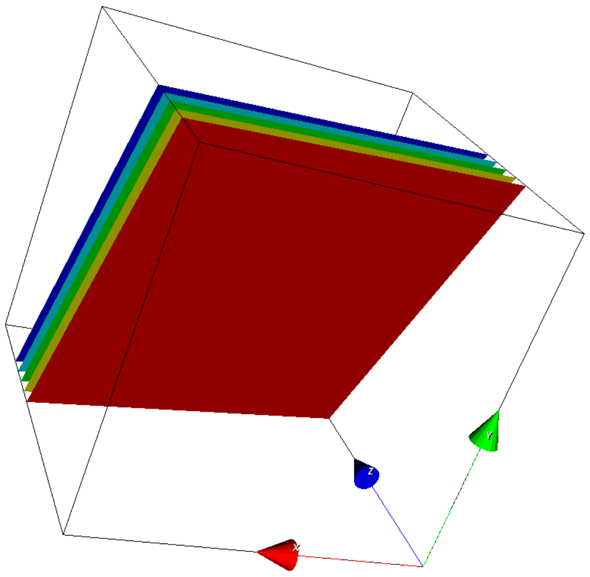

Figure 2b: Function 2

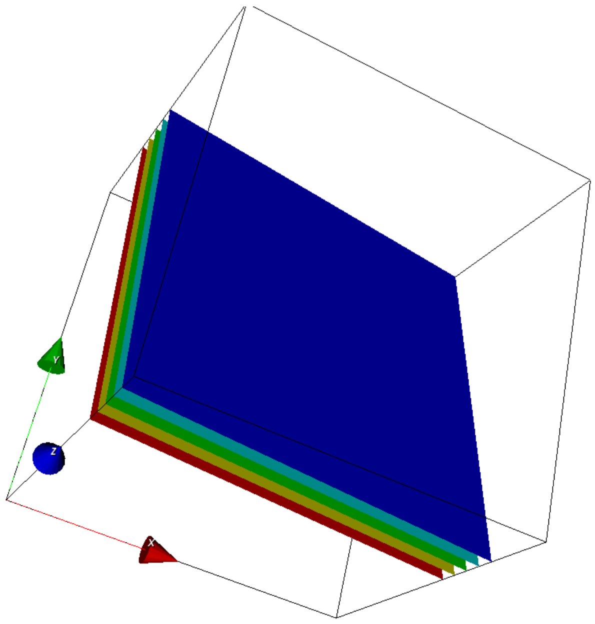

Figure 2c: Function 2

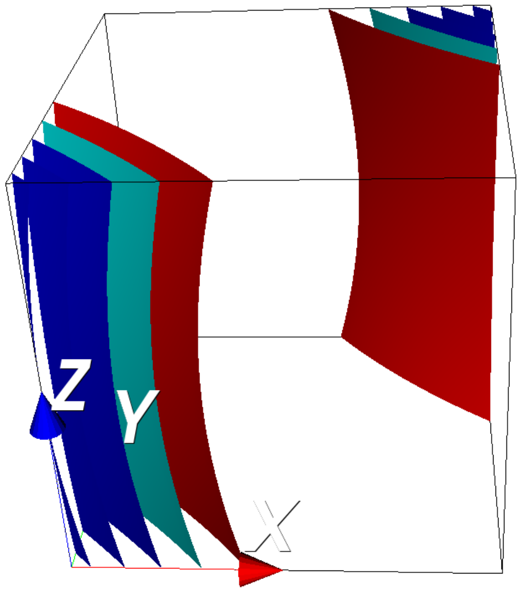

Figure 3a: Function 3

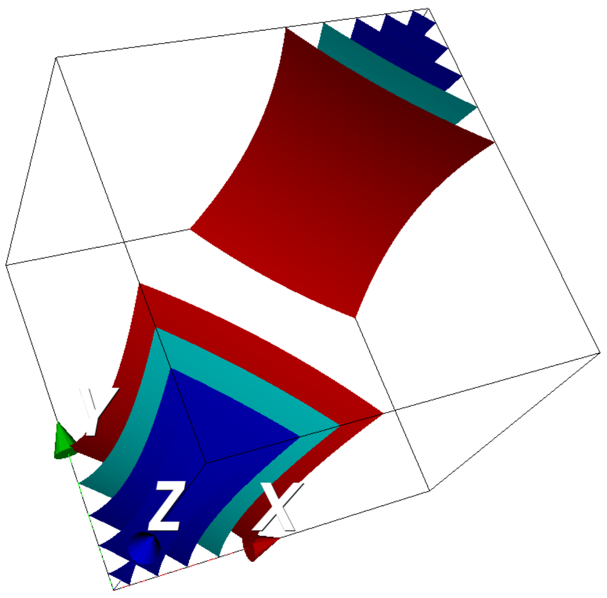

Figure 3b: Function 3

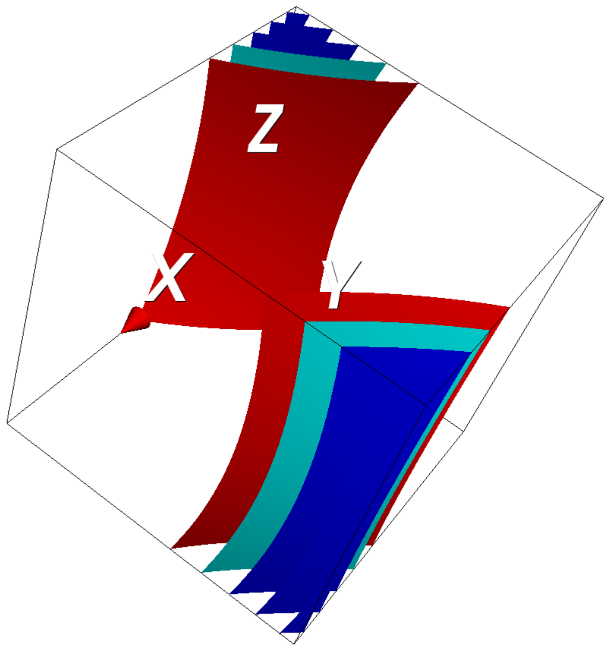

Figure 3c: Function 3

4 Shading

Flat shading only apply one equation to one normal of a polygon. So for a square polygon of 16×16 pixels, the complexity is $\Theta(1)$.

Gourand shading apply equations to the vertices. A square polygon has 4 vertices, so this step need $\Theta(4)$. Then grourand shading uses scan-line interpolation to fill the edge and interior points. Let's say the square is filled in up to bottom, then we need to fill 16 lines. So this step need $\Theta(16)$. So the total is $\Theta(4) + \Theta(16)$.

Phong shading also apply equations to the vertices. So for a square it is $\Theta(4)$. Then is use interpolation for each point of interior and edge. So for a 16×16 square we need $\Theta(256)$. So the total is $\Theta(4) + \Theta(256)$.

5 Example and modification

The example I choose is to visulize the elevation on a plane. At first some grid of points are generated with slightly shift on x, y, z coordinate. So the data are not so regular and with different elevation on the z direction. Then the data is sent to vtkPolyData. Then the data is sent to vtkDelaunay2D to separate the whole plane as many triangles. At here vtkDelaunay2D is used. In the default setting of vtkDelaunay2D, the triangles are on the $x-y$ plane and z coordinate is ignored. Now the output of vtkDelaunay2D is vtkPolyData, which contain the coordinates of the points and the triangle relations of each other.

The next step is find out the maximum and minimum value of the z direction of the data. Then a color lookup table is built and set the range between maximum and minimum. Then a for loop is used to find out the color of each point in the data via the color lookup table. Then all the color of all points are insert to the vtkPolyData of output of vtkDelaunay2D. So now the final vtkPolyData contain three type of elements for a point: postion, triangle relations, and color.

Then the rest is the normal steps such as mapping, renderer for showing the data. The result is Figure 4a.

Now I plan to roll the same plane as a cylinder.

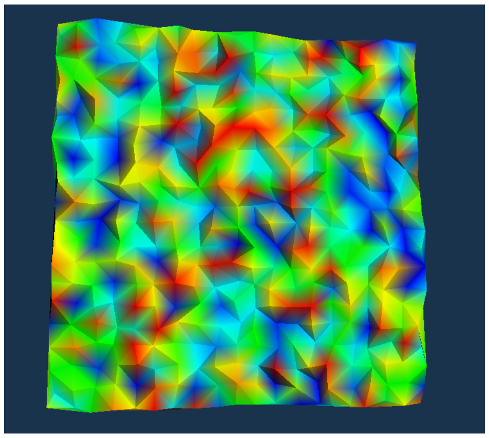

Figure 4a: A plane with rill

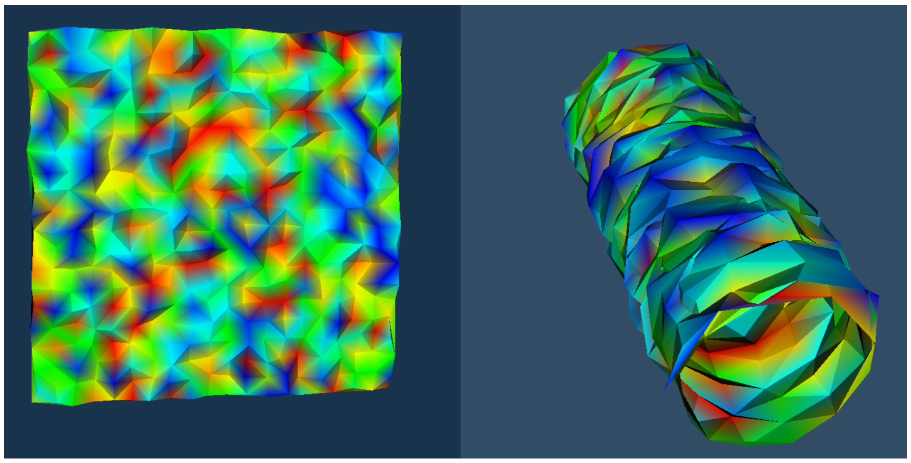

Figure 4b: Plane and Cylinder

At beginning the idea is follow the same step of the example: generate grid points of a cylinder surface, then through vtkDelaunay2D to triangulate, then the same steps of color lookup table, and so on. But the problem at here is vtkDelaunay2D is on $x - y$ plane and ignore the z direction. So if we generate a wrinkled cylinder surface, and use vtkDelaunay2D, it will treat the whole surface as a plane. So it just like squash the cylinder and triangulate it.

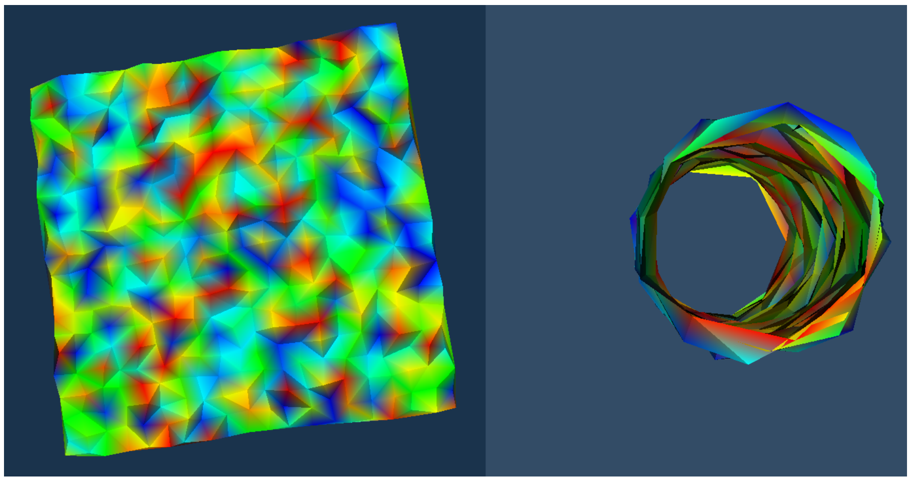

Figure 4c: Plane and Cylinder

Go back to think about the original example. Compare with the original example, what we want to do is have the same triangle relations, but with different positions of points. So for the three elements of the vtkPolyData before mapper, postion, triangle relations, and color, we want to keep the last two and change the first one.

So the idea is, map the point in the plane to a cylinder, and use the new position to replace the original position in the vtkPolyData. We should find out the mapping between the plane and the cylinder. It's clearly one coordinate of $x - y$ will become to z. Let's say it is y. Then the shift of y now is the shift of z. x now is arc. So from the arc we can get the angles of the points, and use $sin()$ and $cos()$ to find out the new x and y. So now we have the new position of the points on the cylinder surface. Then we can use it to replace the original position. In the implementation I want to show the plane and the cylinder at the same time, so I use DeepCopy() to get an identical vtkPolyData, and then do the replacement. The result is shown in Figure 4b and 4c.