# High Dimensional Function Visualization with ParaView Documentation

COSC 3307

Hongrui Guo

## 1. Cost Functions

In this example, a few cost functions are used including test functions, CSRBF, and toy protein model. They are implemented in python as following:

4 Test functions:

```
   def StyblinskiTang(x):
       (n,d) = x.shape
       return np.asarray([[np.sum(np.array([0.5*(x[i,j]**4 -
16*x[i,j]**2 + 5*x[i,j]) for j in range(d)])) for i in range(n)]])

   def Rosenbrock(x):
   (n,d) = x.shape
       return np.asarray([[np.sum(np.array([(100.0 * (x[i,j+1] -
x[i,j]**2)**2 + (1 - x[i,j])**2) for j in range(d-1)])) for i in
range(n)]]).T

   def Rastrigin(x):
       (n,d) = x.shape
       return np.asarray([[np.sum(np.array([10+(x[i,j]**2 -
10*cos(2*pi*x[i,j])) for j in range(d)])) for i in range(n)]]).T

   def Sphere(x):
       (n,d) = x.shape
       return np.asarray([[np.sum(np.array([x[i,j]**2 for j in
range(d)])) for i in range(n)]]).T
```

CSRBF including a class to store parameters, a function to compute parameter using points on the true cost function, and a function for CSRBF itself:

```
   class CSRBF_struct():
       Nn = 0; Nd = 0  # num of points; dimension
       a = 0           # scaling factor
       x = None        # points on surrogate model
       lam = None      # lambda for the surrogate model

   def CSRBF_3_1 (V, st):
       npt = V.shape[0]
       fx = np.zeros((npt, 1))
       for ip in range(npt):
           s = 0.0
           for i in range (st.Nn): # (i = 0; i < st->Nn; i++) {
               r = 0.0;
               for j in range(st.Nd): # (j = 0; j < st->Nd; j++) {
                   r += (V[ip, j] - st.x[i, j])**2
               r = sqrt(r) / st.a

               if r <= 1.0:
                   temp0 = (1.0 - r) ** 4.0
                   temp1 = (4.0 * r) + 1.0
                   s += st.lam[i] * (temp0 * temp1)
           fx[ip, 0] = s
```

```
        if ip%1000 == 0:
            print(ip,"/", npt)
    return fx

def setup_csrbf(X, a, f):
    N, D = X.shape
    z = f(X)
    R = np.zeros((N, N))
    for i in range(N):
        for j in range(N):
            R[i, j] = np.linalg.norm(X[i, :] - X[j, :]);
    R /= a
    PHI = ((1 - R)**4) * (4*R + 1)
    PHI *= (R < 1.0)
    lam = np.linalg.lstsq(PHI, z.flatten())[0] # lam = PHI\z

    params = CSRBF_struct()
    params.Nn = N; params.Nd = D; params.a = a
    params.x = X; params.lam = lam
    return params
```

Toy protein model:

```
def Protein(M, TH):
    Marray = [ord(m)-ord('A') for m in list(M)]
    n = len(M); npt = TH.shape[0]
    TH = np.append(np.zeros((npt,1)), TH, axis=1)
    PHI = np.zeros((npt,1))
    C = [[1.0, -1/2], [-1/2, 1/2]]

    for ip in range(npt):
        th = TH[ip, :]
        V1 = sum([0.25 * (1 - cos(x)) for x in th[1:n-1]])
        V2 = 0.0

        for i in range(0, n-2):
            for j in range(i+2, n):
                ii = Marray[i]
                jj = Marray[j]

                rij = 0.0
                t1 = 0.0
                t2 = 0.0
                for k in range(i+1, j):
                    thtemp = sum(th[i+1:k+1])
                    t1 = t1 + cos(thtemp)
                    t2 = t2 + sin(thtemp)
                rij = sqrt((1.0 + t1)**2 + t2**2)

                V2 = V2 + (4.0 * (rij**-12.0 - C[ii][jj]*rij**(-
6.0)))
        PHI[ip, 0] = V1 + V2;
    return PHI
```

## 2. 3D Grid and Point Cloud

To visuals a 3D function, a few 3D points can be easily send in the function and a point cloud can be formed from the points and values returned. However, for functions with dimension $d$ higher than 3, a 3D slice needs to be taken from the $D$-D function for visualization. To do so, $D - 3$ of the dimensions need to be fixed at certain values.

Take CSRBF for a 4D sphere as an example. First, the dimension, number of points on surrogate, upper/lower bound, the scaling factor, and the true cost function is defined.

```
n_rbf = 1000; D = 4
lb = -5; ub = 5
a = 10; f = Sphere
```

Generate `n_rbf` random $D$-D points `X` in the domain and compute parameters for the CSRBF formed with points `X`.

```
X = np.random.uniform(low=lb, high=ub, size=(n_rbf,d))
params = setup_csrbf(X, a, f)
```

Define the resolution of the grid and where the fixed dimensions. `np.nan` indicates which axes are not fixed. In this case, the 3D grid is 30×30×30 and the 4th dimension is fixed at 0.0.

```
Ngrid=30; fixed = np.asarray([np.nan, np.nan, np.nan, 0.0])
```

Get dimensions where the slice is not taken along and the dimension of the slice. In this case, it is a 3D slice taken along the 4th axis (`d_indx` = [0,1,2]) at 0.0.

```
d_indx = np.asarray(np.where(np.isnan(fixed))).reshape((-1,))
d_grid = len(d_indx)
```

Set up a grid of points, evaluate $f_{RBF}(x)$ using the grid and fixed value, and fill an $Ngrid^{d\_grid} \times D + 1$ array with values $f_{RBF}(x)$ in the 1st column and coordinates $x$ in the rest columns.

```
GRID = np.linspace(xmin, xmax, Ngrid)
N = Ngrid**d_grid
pts = np.zeros((N, D+1))
x = np.zeros(D)
Ngrid_to_i = Ngrid ** np.arange(0, d_grid)

for ii in range(0, N):
    for i in range(0, D):
        if i in d_indx:
            indx = int(ii / Ngrid_to_i[i]) % Ngrid
            x[i] = GRID[indx]
        else:
            x[i] = fixed[i]
    pts[ii][1:D+1] = x

pts[:, 0] = CSRBF_3_1(pts[:, 1:D+1], params).flatten()
```

Taking a 3D slice from other cost functions is identical to the above example.

3. Save Point Cloud as VTK Grid file

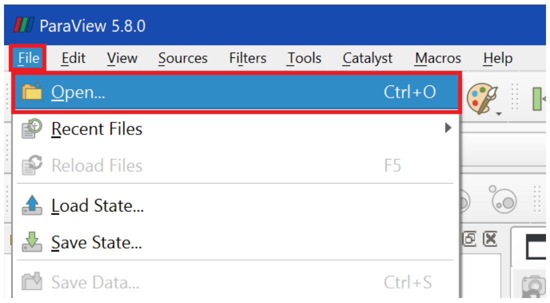
To visualize the 3D slice, the coordinates of the grid and the values need to be saved as a point cloud in VTK UnstructuredGrid (.vtu) file format. To do so, the PyEVTK library need to be installed by downloading the source code and running the `setup.py`. To save the points, only the value and coordinates for the grid are needed.

```
from evtk.hl import pointsToVTK
name = f.__name__+"_CSRBF"
name += "_"+str(D)+"D_"+str(Ngrid)+"ptsGrid_"+str(list(fixed))
name += "_"+str(n_rbf)+"ptsRBF"
pointsToVTK(name, np.ascontiguousarray(pts[:, d_indx[0]+1]),
          np.ascontiguousarray(pts[:, d_indx[1]+1]),
          np.ascontiguousarray(pts[:, d_indx[2]+1]),
          data = {"fx" : np.ascontiguousarray(pts[:, 0].flatten())})
```

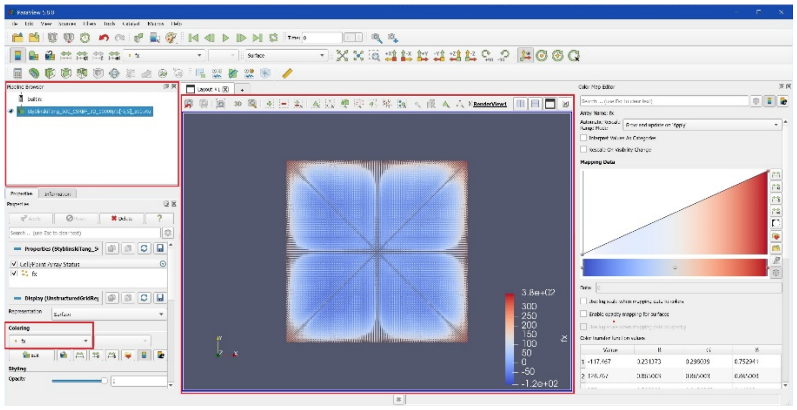
4. ParaView Visualization Pipeline

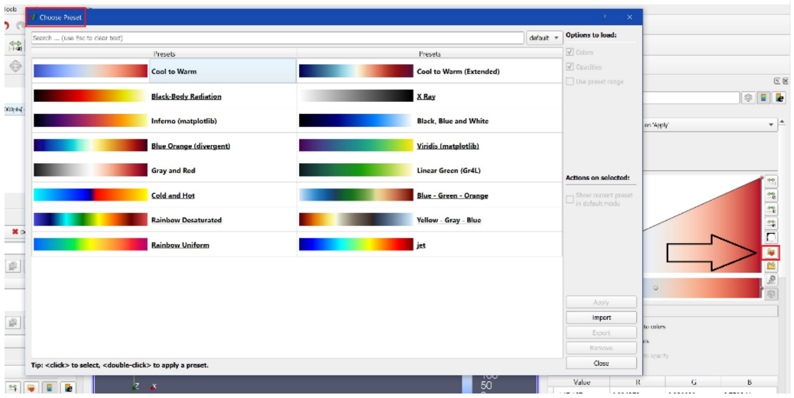
To visualize point clouds, ParaView 5.8.0 desktop client is used. A 3D Styblinski-Tang function is used to demonstrate ParaView visualization pipeline and some of the filters.

First, open the vtu file for the point cloud.

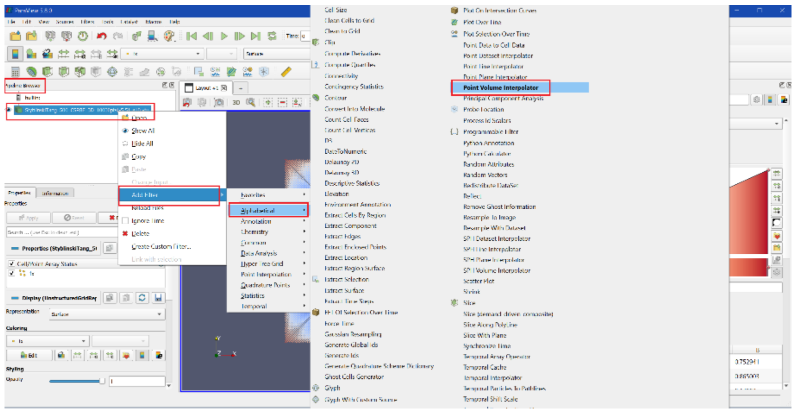


Then the file will show up in the *pipeline browser*, and a cube-shaped white point cloud will be shown in the *render view*. Change the *coloring* from *solid color* to *fx* to display the value with scaled colors. The theme of the color can also be changed to a different preset.

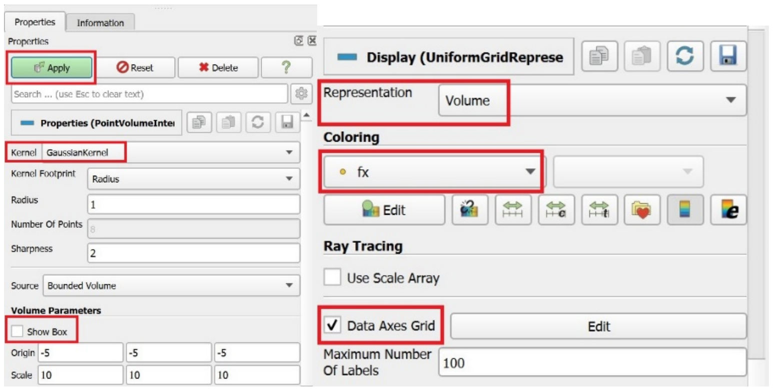




Right-click the file in the *pipeline browser* to add a *Point Volume Interpolator* filter to generate a 3D volume from the point cloud.



Then change the *Kernel* to *GaussianKernel*, uncheck *Show Box*, change *Representation* to *Volume*, change *Coloring* to `fx`, check *Data Axes Grid*, and finally click *Apply*. Left-click and drag to rotate the visualization and use the scroll wheel to zoom in and out.



Properties | Information

Properties (PointVolumeInte)

| | |
|---|---|
| Kernel | GaussianKernel |
| Kernel Footprint | Radius |
| Radius | 1 |
| Number Of Points | 8 |
| Sharpness | 2 |
| Source | Bounded Volume |

Volume Parameters

- [ ] Show Box

| | | | |
|---|---|---|---|
| Origin | -5 | -5 | -5 |
| Scale | 10 | 10 | 10 |

Display (UniformGridReprese

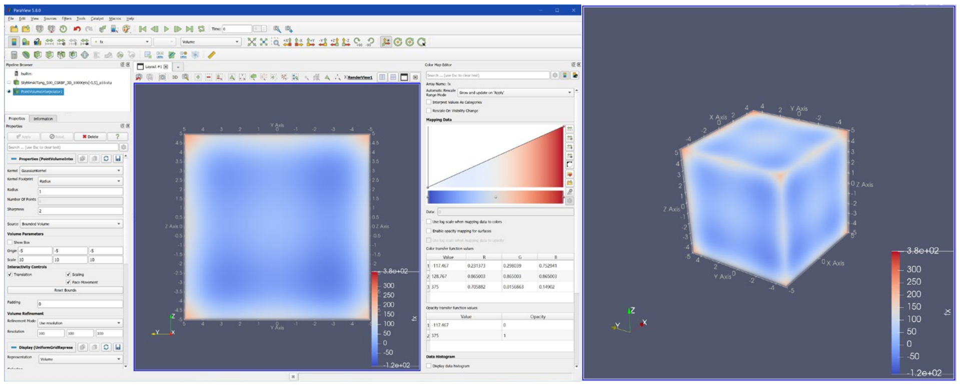
Representation: Volume

Coloring: fx

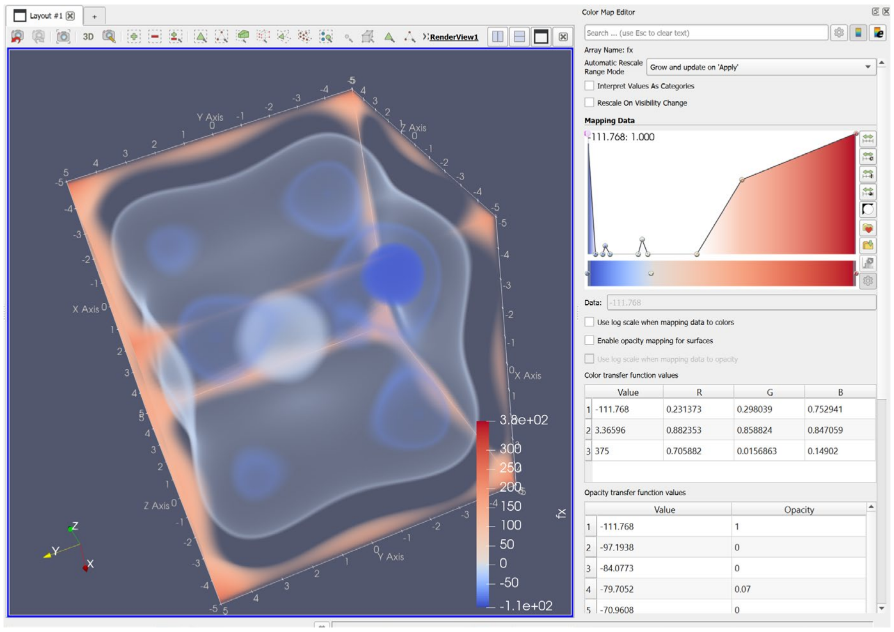
Ray Tracing

- [ ] Use Scale Array
- [✓] Data Axes Grid

Maximum Number Of Labels: 100



The color map can then be modified to give better visuals. The x-axis is for colors and the y-axis is for transparency.



Color Map Editor

Array Name: fx

Automatic Rescale Range Mode: Grow and update on 'Apply'

- [ ] Interpret Values As Categories
- [ ] Rescale On Visibility Change

Mapping Data

-111.768: 1.000

Data: -111.768

- [ ] Use log scale when mapping data to colors
- [ ] Enable opacity mapping for surfaces
- [ ] Use log scale when mapping data to opacity

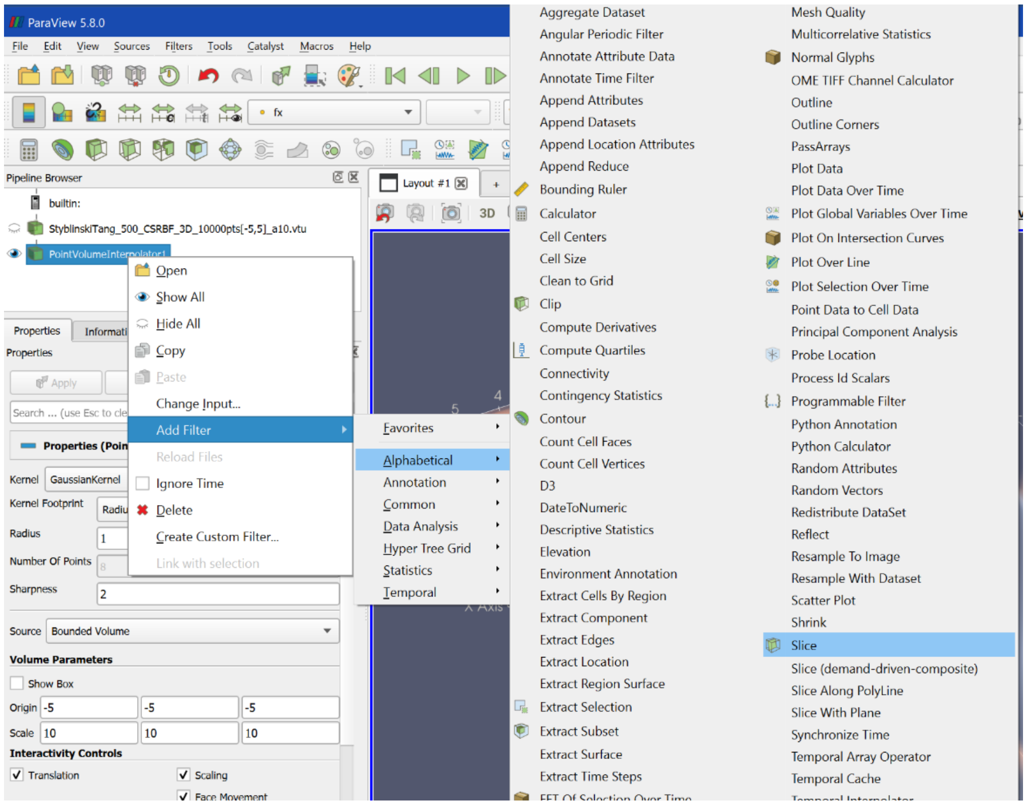
Color transfer function values

| | Value | R | G | B |
|---|---|---|---|---|
| 1 | -111.768 | 0.231373 | 0.298039 | 0.752941 |
| 2 | 3.36596 | 0.882353 | 0.858824 | 0.847059 |
| 3 | 375 | 0.705882 | 0.0156863 | 0.14902 |

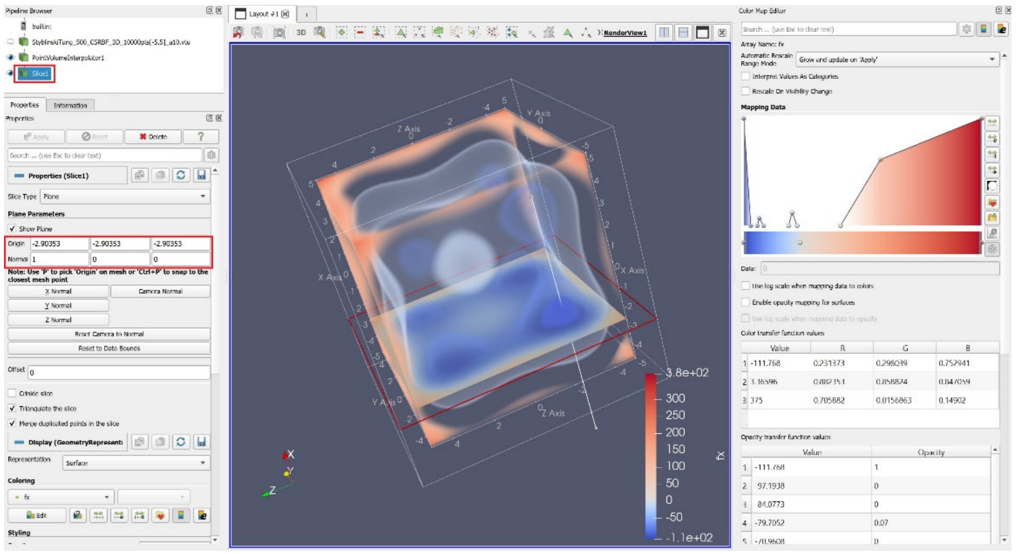
Opacity transfer function values

| | Value | Opacity |
|---|---|---|
| 1 | -111.768 | 1 |
| 2 | -97.1938 | 0 |
| 3 | -84.0773 | 0 |
| 4 | -79.7052 | 0.07 |
| 5 | -70.9608 | 0 |

A 2D slice can be taken from the 3D volume by adding a *slice* filter. The origin of the plane is set to its global minimum at (-2.903534, -2.903534, -2.903534)



Properties (PointVolumeInterpolator1)
Kernel: GaussianKernel
Radius: 1
Number Of Points: 8
Sharpness: 2
Source: Bounded Volume
Volume Parameters
Show Box: ☐
Origin: -5, -5, -5
Scale: 10, 10, 10
Interactivity Controls
Translation: ✓
Scaling: ✓
Face Movement: ✓



Properties (Slice1)
Slice Type: Plane
Plane Parameters
Show Plane: ✓
Origin: -2.90353, -2.90353, -2.90353
Normal: 1, 0, 0
Note: Use 'P' to pick 'Origin' on mesh or 'Ctrl+P' to snap to the closest mesh point
Offset: 0
Crinkle slice: ☐
Triangulate the slice: ✓
Merge duplicated points in the slice: ✓
Display (GeometryRepresentation)
Representation: Surface
Coloring: fx
Styling

Array Name: fx
Automatic Rescale Range Mode: Grow and update on 'Apply'
Interpret Values As Categories: ☐
Rescale On Visibility Change: ☐
Data: 0
Use log scale when mapping data to colors: ☐
Enable opacity mapping for surfaces: ☐
Use log scale when mapping data to opacity: ☐

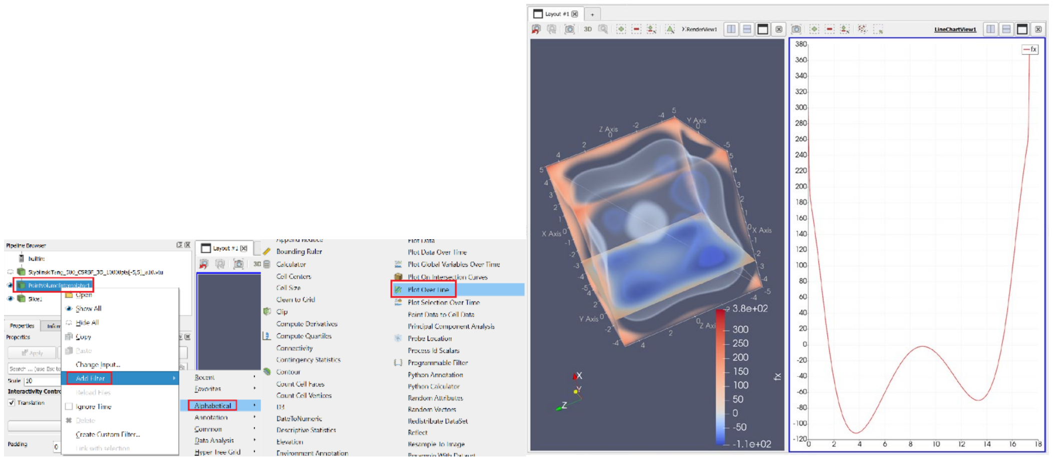
Color transfer function values

| | Value | R | G | B |
|---|---|---|---|---|
| 1 | -111.768 | 0.231373 | 0.298039 | 0.752941 |
| 2 | 131.6596 | 0.882353 | 0.858824 | 0.847059 |
| 3 | 375 | 0.705882 | 0.0156863 | 0.14902 |

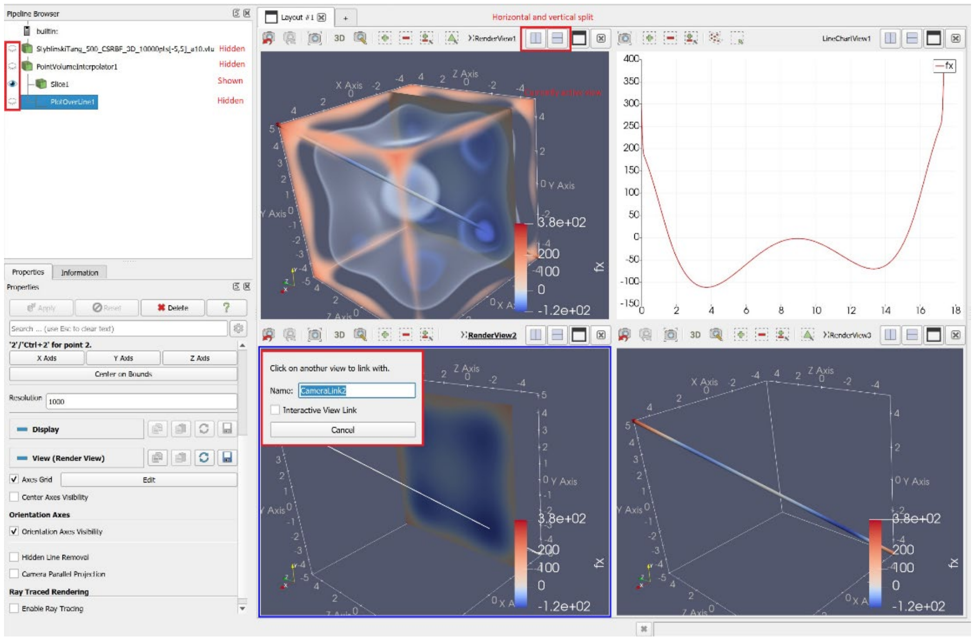
Opacity transfer function values

| | Value | Opacity |
|---|---|---|
| 1 | -111.768 | 1 |
| 2 | 97.1938 | 0 |
| 3 | 84.0773 | 0 |
| 4 | -79.7052 | 0.07 |
| 5 | -70.9608 | 0 |

*Plot Over Line* filter can be added to plot the values on a line segment. After it is added and applied, a new view an automatically opened to display the plot. *Representation* can be changed to *Points* for thicker line in the render view. In this example, the line segment goes through 1 local minimum (light blue), 1 global minimum (dark blue), and 1 local maximum (bluish white).

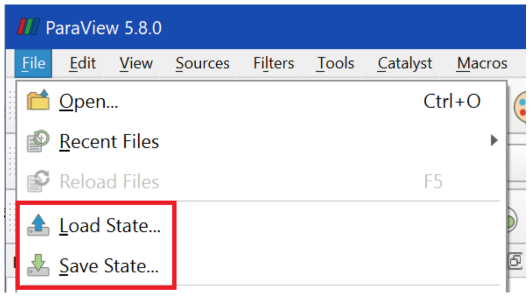


The blue outline around the view indicates it is active. The view can also be split manually, and the objects can be hidden or shown independently in different views. Each view can be renamed by right-clicking its title. The camera for each view can also be linked by right-clicking a view then choose *link camera* to choose the source camera and left click another to select another.

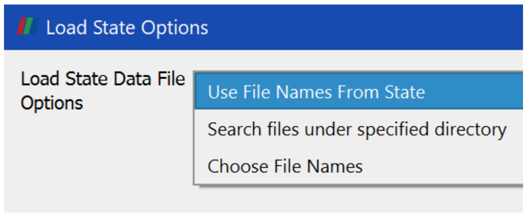


Click on another view to link with.
Name: CameraLink2
Interactive View Link
Cancel

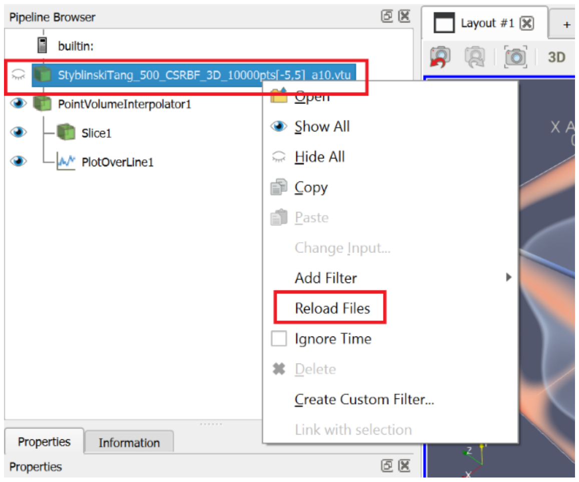
The visualization pipeline including filters, files, color maps, and everything else in the current session can be saved and loaded as a state in pvsm file format.



Each time the state is loaded the input file can be chosen to be the original or different ones, so the pipeline can be reused on different data.

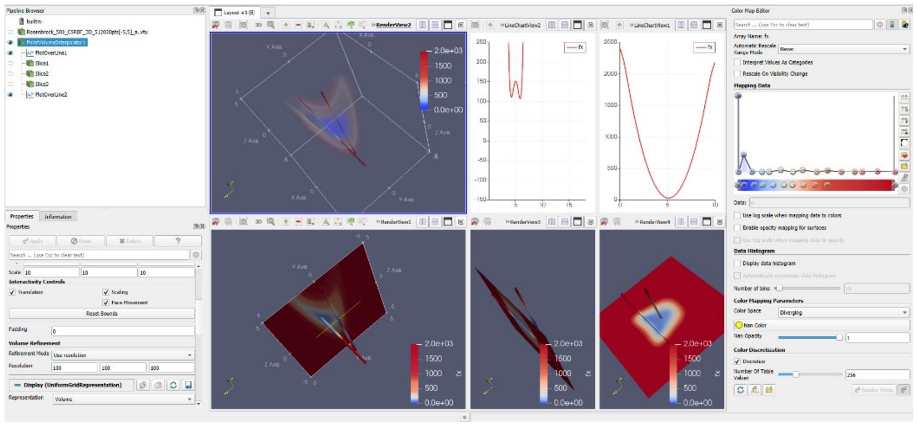

Load State Options
Load State Data File Options
Use File Names From State
Search files under specified directory
Choose File Names

If the data in the files are changed, the files can be reloaded directly.

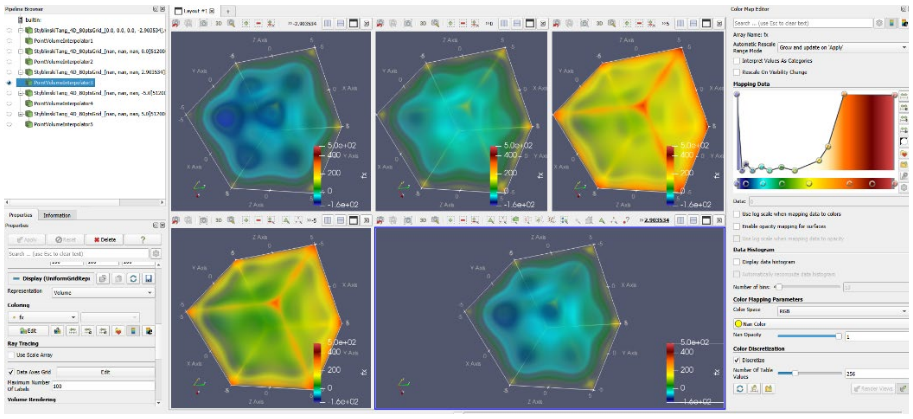


More examples on ParaView visualization

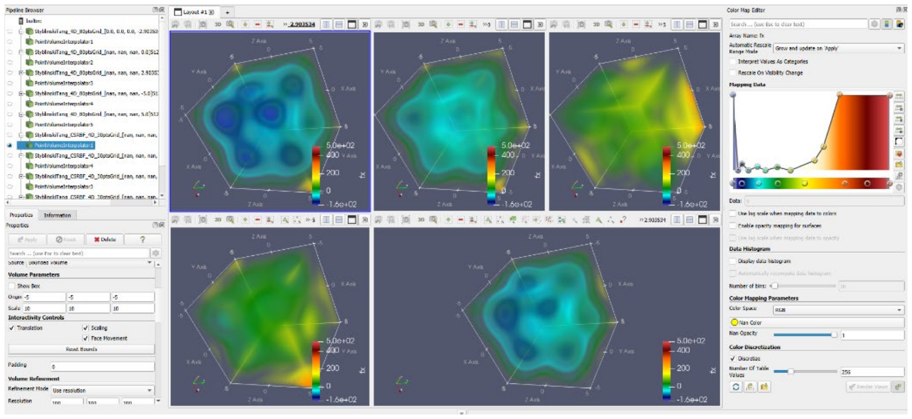
3D Rosenbrock



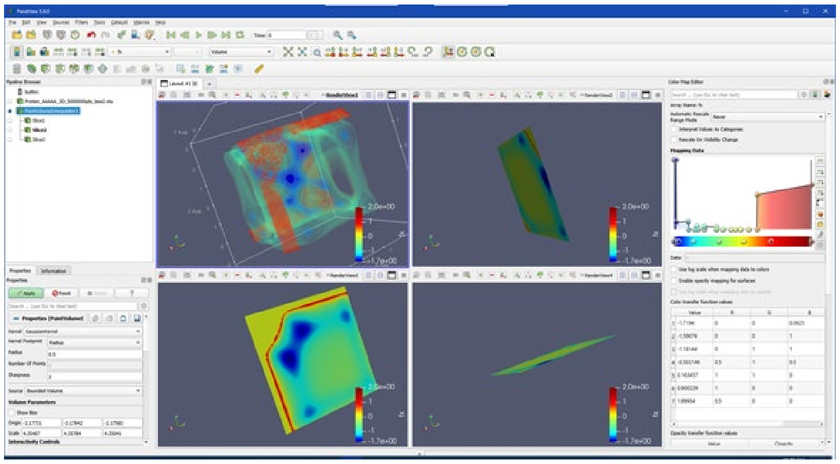
4D Styblinski-Tang, 4th dimension fixed at -5, -2.903534, 0, 2.903534, 5



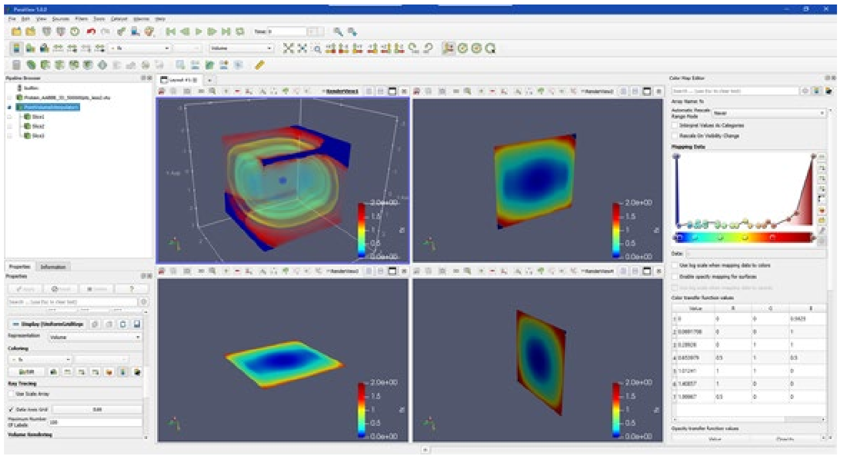
CSRBF of 4D Styblinski-Tang, a = 10, n = 1000, 4th dimension is fixed at the same dimension as above



Toy protein AAAAA



Toy protein AABBB



Toy protein ABABAB 4th angle fixed at -2, -1, 0, 1, 2. Potential minima are circled in red.