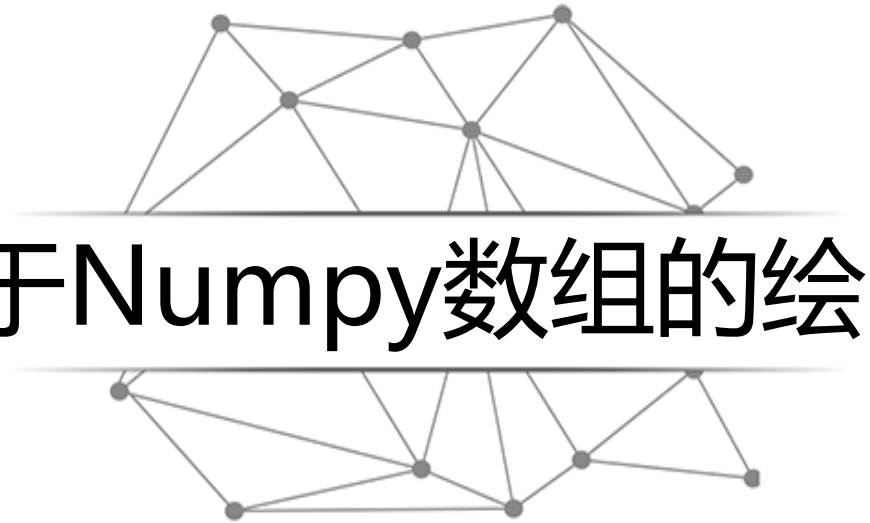


基于Numpy数组的绘图函数

mlab对Numpy建立可视化过程：

1、建立数据源

2、使用Filter（可选）

3、添加可视化模块

3D绘图函数-0D和1D数据

函数	说明
Point3d( )	基于Numpy数组x、y、z提供的三维点坐标，绘制点图形
Plot3d( )	基于1维Numpy数组x、y、z提供的三维坐标数据，绘制线图形

3D绘图函数-Points3d()

函数形式：

```
points3d(x, y, z…)
points3d(x, y, z, s, …)
points3d(x, y, z, f, …)
```

x,y,z表示numpy数组、列表或者其他形式的点三维坐标

s表示在该坐标点处的标量值

f表示通过函数f（x，y，z）返回的标量值

3D绘图函数-Points3d ()

参数	说明
color	VTK对象的颜色，定义为(0,1)的三元组
colormap	colormap的类型，例如Reds、Blues、Copper等
extent	x、y、z数组范围[xmin, xmax, ymin, ymax, zmin, zmax]
figure	画图
line_width	线的宽度，该值为float，默认为0.2
mask_points	减少/降低大规模点数据集的数量
mode	符号的模式，例如2darrow、2dcircle、arrow、cone等
name	VTK对象名字

3D绘图函数-Points3d ()

参数	说明
opcity	Vtk对象的整体透明度，该值为float型，默认为1.0
reset_zoom	对新加入场景数据的放缩进行重置。默认为True
resolution	符号的分辨率，如球体的细分数，该值为整型，默认为8
scale_factor	符号放缩的比例
scale_mode	符号的放缩模式，如vector、scalar、none
transparent	根据标量值确定actor的透明度
vmax	对colormap放缩的最大值
vmin	对colormap放缩的最小值

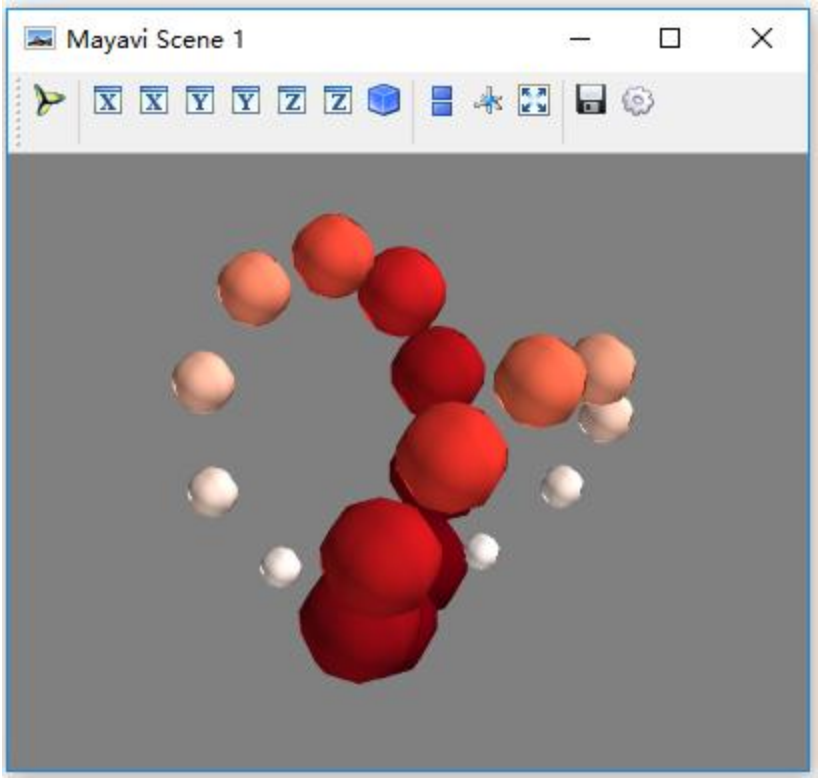
3D绘图函数-Points3d ()

```
import numpy as np
from mayavi import mlab

#建立数据
t = np.linspace(0, 4 * np.pi, 20)
x = np.sin(2 * t)
y = np.cos(t)
z = np.cos(2 * t)
s = 2 + np.sin(t)

#对数据进行可视化
points = mlab.points3d(x, y, z, s, colormap="Reds", scale_factor=.25)
mlab.show()
```

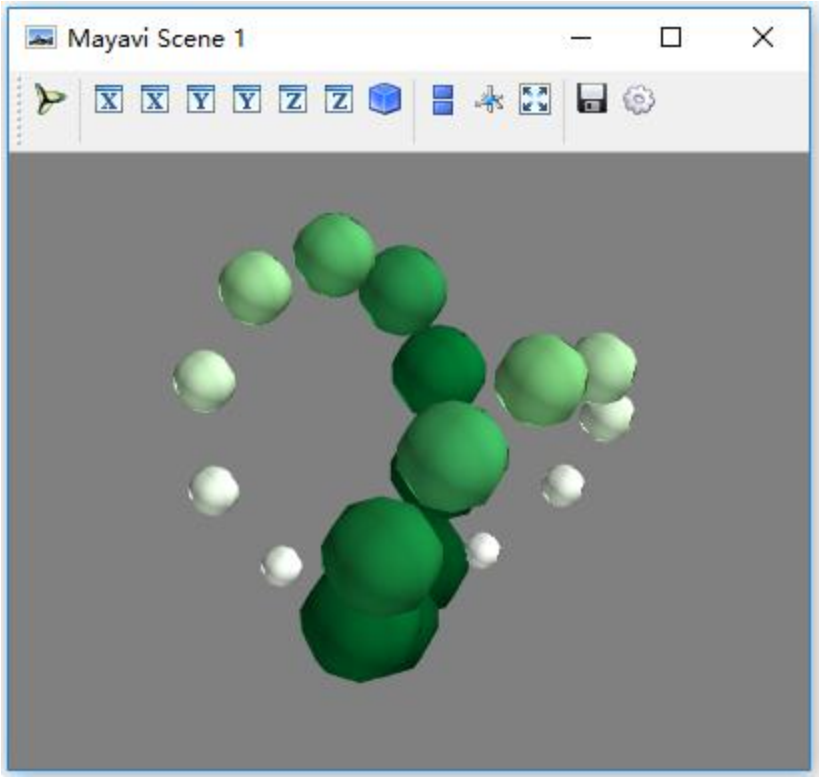
3D绘图函数-Points3d ()

```
points3d(x, y, z, s, colormap="Reds", scale_factor=.25)
```

3D绘图函数-Points3d ()



3D绘图函数-Points3d ()



mayavi.mlab.show()

```
mayavi.mlab.show(func = None, stop = False)
```

3D绘图函数-plot3d ()

函数形式：

```
plot3d(x, y, z…)
plot3d(x, y, z, s, …)
```

x,y,z表示numpy数组，或列表。给出了线上连续的点的位置

s表示在该坐标点处的标量值

3D绘图函数-plot3d ()

color、colormap、extent、figure、line_width、name、opacity、representation、reset_zoom、transparent、tube_radius、tube_sides、vmax、vmin

参数	说明
tube_radius	线管的半径，用于描述线的粗细
tube_sides	表示线的分段数，该值为整数，默认为6

3D绘图函数-plot3d ()

```
import numpy as np
from mayavi import mlab

#建立数据
n_mer, n_long = 6, 11
dphi = np.pi / 1000.0
phi = np.arange(0.0, 2 * np.pi + 0.5 * dphi, dphi)
mu = phi * n_mer
x = np.cos(mu) * (1 + np.cos(n_long * mu / n_mer) * 0.5)
y = np.sin(mu) * (1 + np.cos(n_long * mu / n_mer) * 0.5)
z = np.sin(n_long * mu / n_mer) * 0.5

#对数据进行可视化
l = mlab.plot3d(x, y, z, np.sin(mu), tube_radius=0.025, colormap='Spectral')
mlab.show()
```

3D绘图函数-plot3d ()

```
plot3d(x, y, z, np.sin(mu), tube_radius=0.025,
                colormap='Spectral')
```

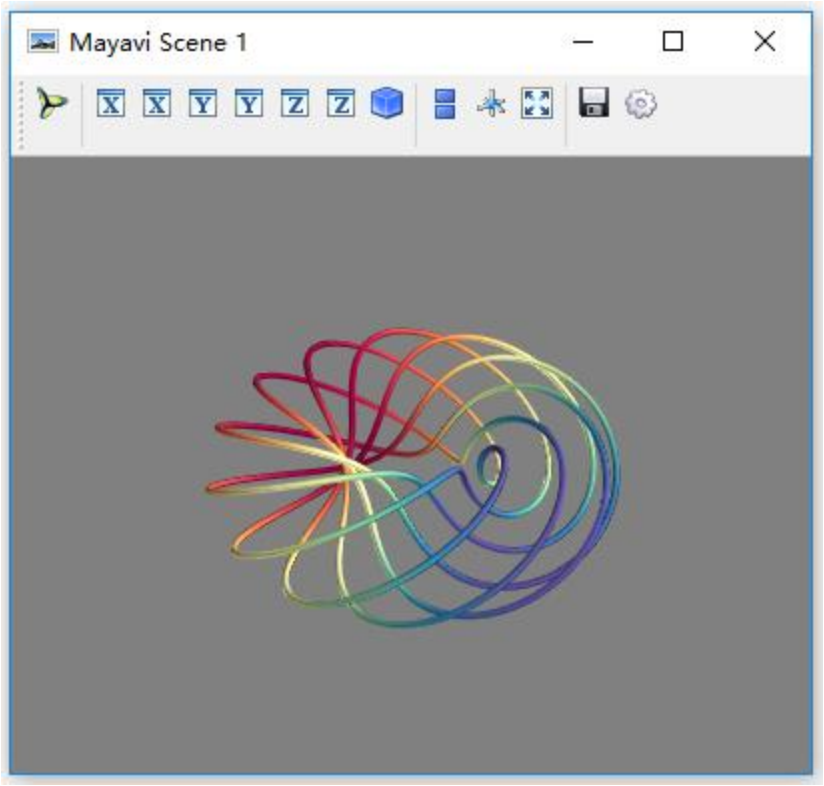
x,y,z表示numpy数组，给出了线上连续的点的位置

np.sin(mu)表示在该坐标点处的标量值

tube_radius绘制线的半径为0.025

colormap采用Spectral颜色模式。

3D绘图函数-plot3d ()



3D绘图函数-2D数据

函数	说明
imshow()	将二维数组可视化为一张图像
surf()	将二维数组可视化为一个平面，z轴描述了数组点的高度
contour_surf()	将二位数组可视化为等高线，高度值由数组点的值来确定
mesh()	绘制由三个二维数组x、y、z描述坐标点的网格平面
barchart()	根据二维、三维或者点云数据绘制的三维柱状图
triangular_mesh()	绘制由x、y、z坐标点描述的三角网格面

3D绘图函数-imshow ()

函数形式：

```
imshow(s, …)
```

s是一个二维数组,s的值使用colormap被映射为颜色

3D绘图函数-imshow ()

color、colormap、extent、figure、interpolate、

line_width、name、opacity、reset_zoom、transparent、

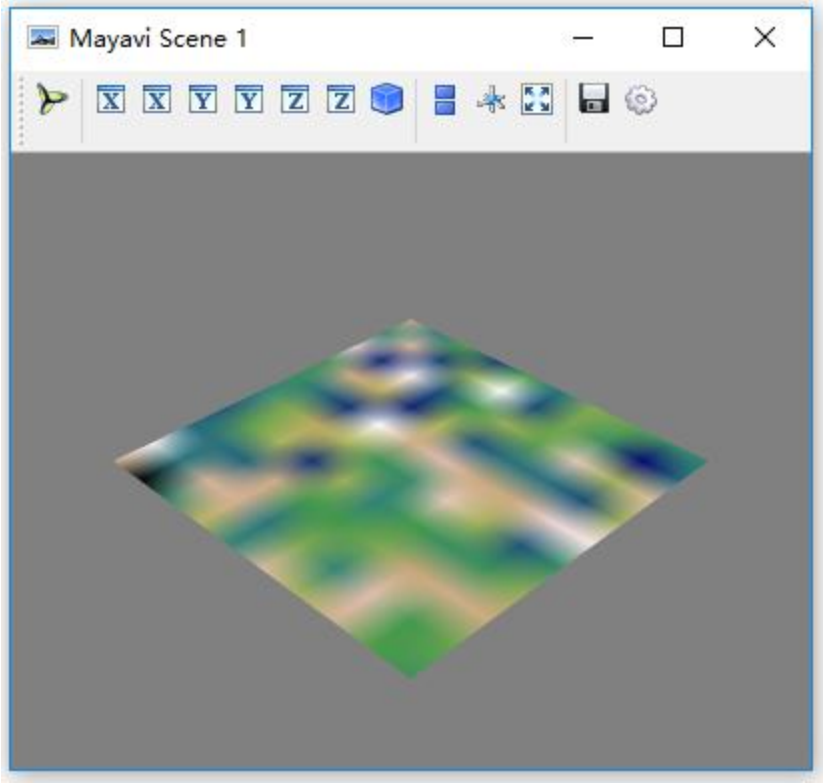
vmax、vmin

参数	说明
interpolate	图像中的像素是否被插值，该值为布尔型，默认为True

3D绘图函数-imshow ()

```
import numpy
from mayavi import mlab
#建立数据
s = numpy.random.random((10, 10))

#对数据进行可视化
img = mlab.imshow(s, colormap='gist_earth')
mlab.show()
```

3D绘图函数-imshow ()



3D绘图函数-surf()

函数形式：

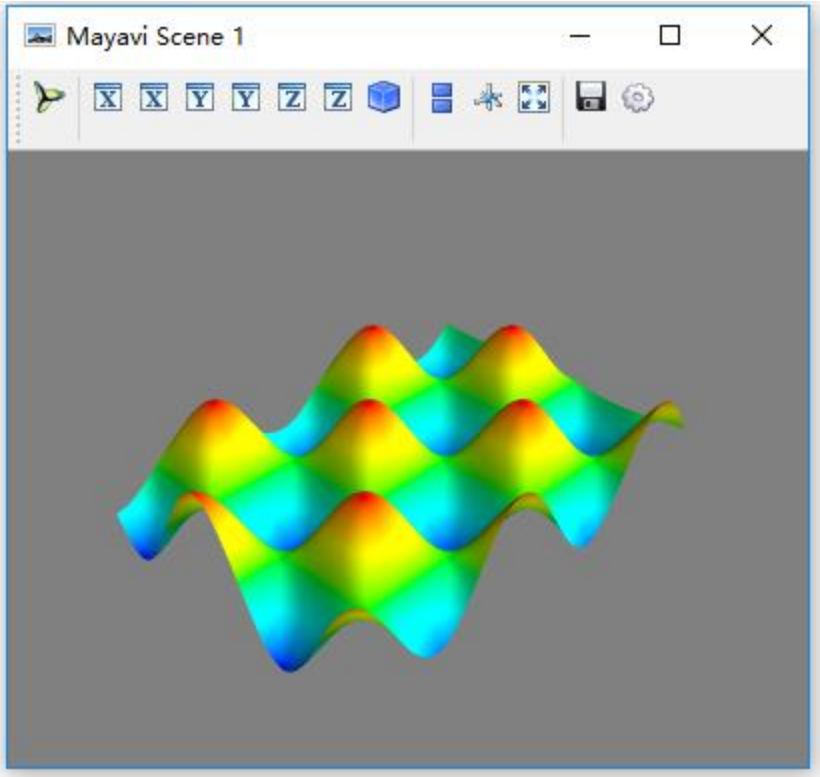
```
surf(s, …)
surf(x, y, s,…)
surf(x, y, f,…)
```

s是一个高程矩阵，用二维数组表示。

3D绘图函数-surf()

```
import numpy as np
from mayavi import mlab

def f(x, y):
    return np.sin(x - y)+np.cos(x + y)

x, y = np.mgrid[-7.:7.05:0.1, -5.:5.05:0.05]
s = mlab.surf(x, y, f)
mlab.show()
```

3D绘图函数-surf()

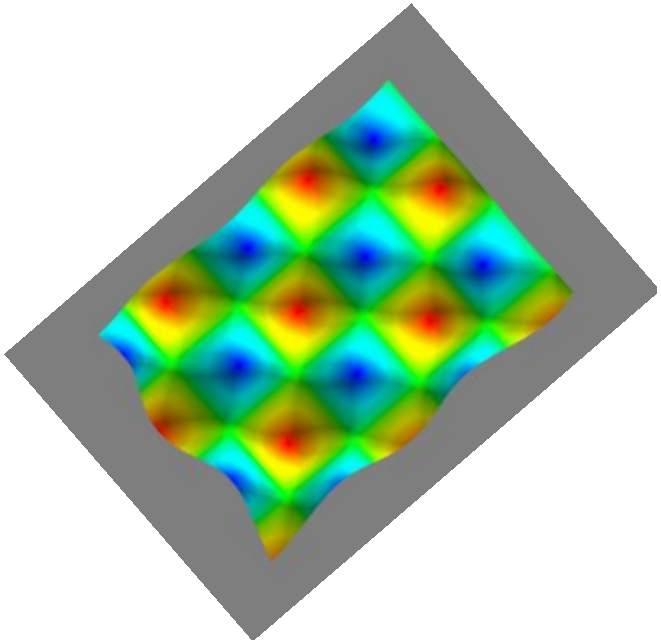


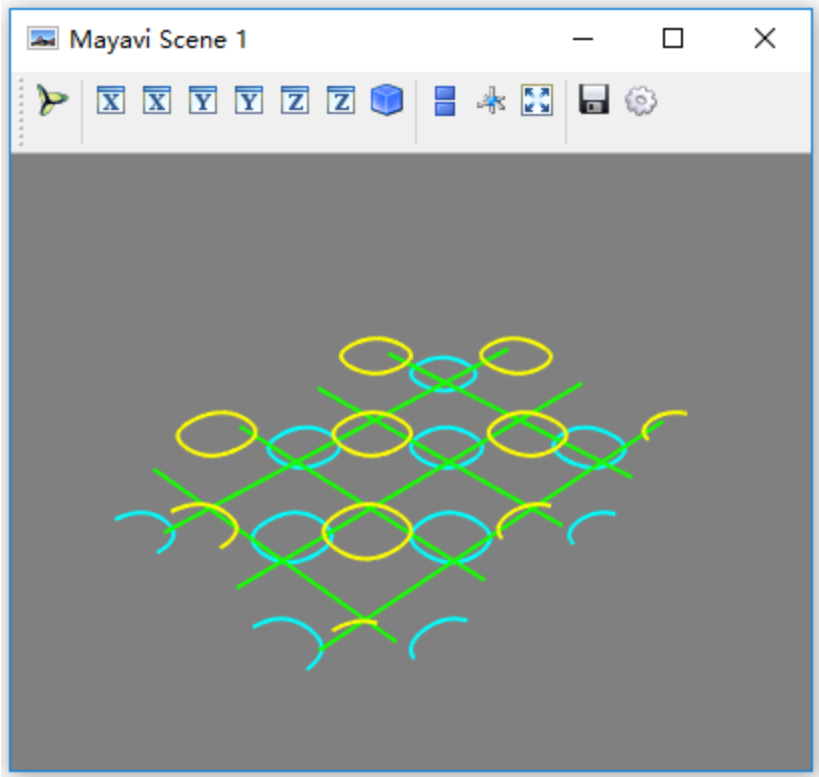
3D绘图函数-contour_surf()

与Surf()类似，但求解的是等值线

```
import numpy as np
from mayavi import mlab

def f(x, y):
    return np.sin(x - y)+np.cos(x + y)

x, y = np.mgrid[-7.:7.05:0.1, -5.:5.05:0.05]
con_s = mlab.contour_surf(x, y, f)
mlab.show()
```

3D绘图函数-contour_surf()





3D绘图函数-3D数据

函数	说明
contour3d()	三维数组定义的体数据的等值面可视化
quiver3d()	三维矢量数据的可视化，箭头表示在该点的矢量数据
flow()	绘制3维数组描述的向量场的粒子轨迹

3D绘图函数-contour3d()

函数形式：

```
contour3d(scalars, …)
contour3d(x, y, z , scalars,…)
```

scalars网格上的数据，用三维numpy数组表示。

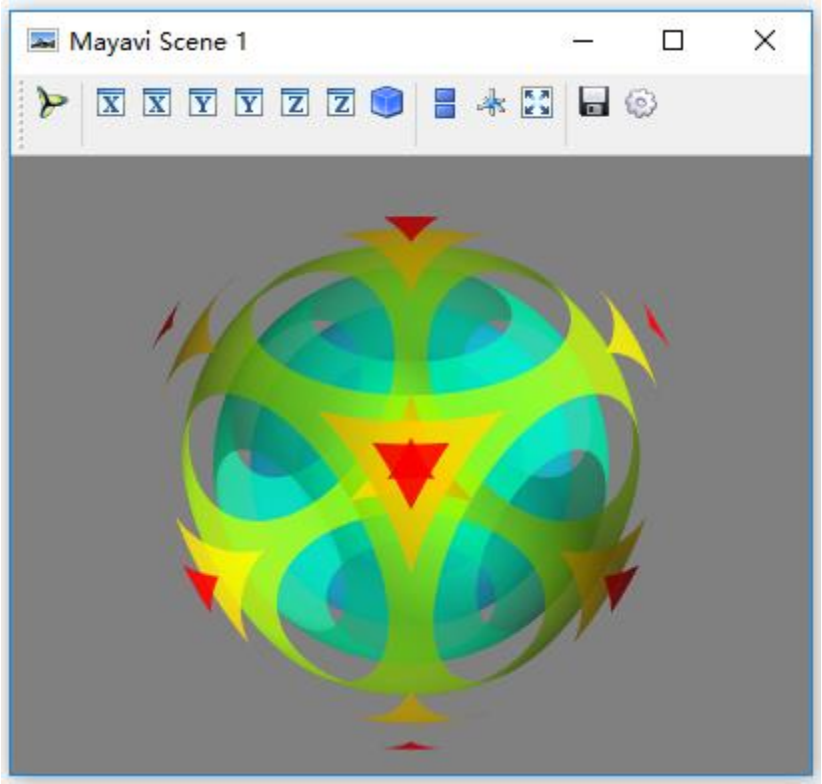
x，y，z三维空间坐标

参数	说明
contours	定义等值面的数量

3D绘图函数-contour3d()

```
import numpy
from mayavi import mlab

x, y, z = numpy.ogrid[-5:5:64j, -5:5:64j, -5:5:64j]
scalars = x * x + y * y + z * z
obj = mlab.contour3d(scalars, contours=8, transparent=True)
mlab.show()
```

3D绘图函数-contour3d()



3D绘图函数-quiver3d()

函数形式：

```
quiver3d(u，v，w …)
quiver3d(x，y，z，u，v，w …)
quiver3d(x，y，z，f，…)
```

u，v，w用numpy数组表示的向量

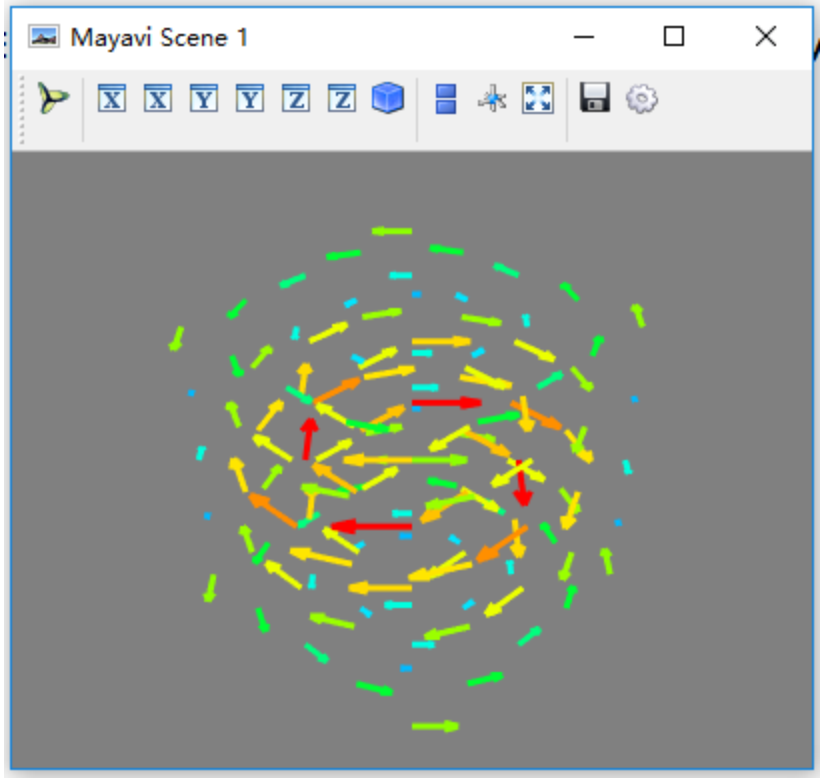
x，y，z表示箭头的位置， u，v，w矢量元素

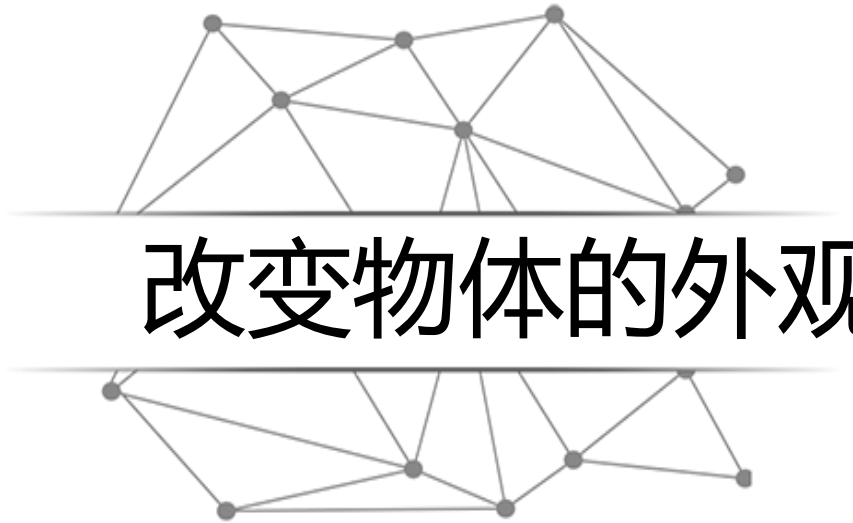
f需要返回在给定位置（x，y，z）的（u，v，w）矢量

3D绘图函数-quiver3d()

```
import numpy as np
from mayavi import mlab

x, y, z = np.mgrid[-2:3, -2:3, -2:3]
r = np.sqrt(x ** 2 + y ** 2 + z ** 4)
u = y * np.sin(r) / (r + 0.001)
v = -x * np.sin(r) / (r + 0.001)
w = np.zeros_like(z)

obj = mlab.quiver3d(x, y, z, u, v, w, line_width=3, scale_factor=1)
mlab.show()
```

3D绘图函数-quiver3d()





改变物体的外观

改变颜色

colormap定义的颜色，也叫LUT。

LUT：Look Up Table。

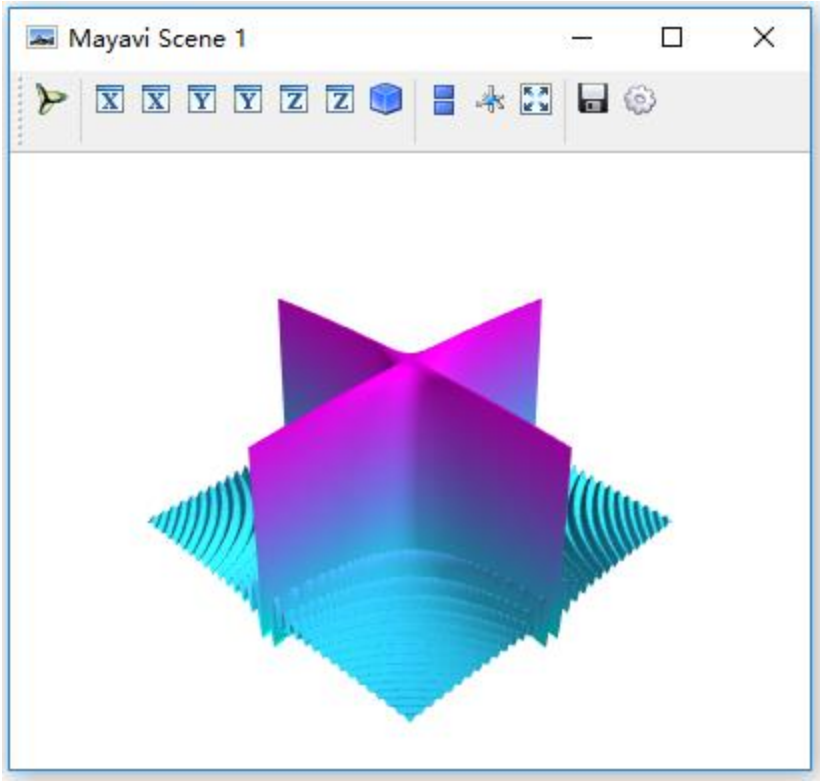
常见的colormaps

=accent	flag	hot	pubu	set2
autumn	gist_earth	hsv	pubugn	set3
black-white	gist_gray	jet	puor	spectral
blue-red	gist_heat	oranges	purd	spring
blues	gist_ncar	orrd	purples	summer
bone	gist_rainbow	paired	rdbu	winter
brbg	gist_stern	pastel1	rdgy	ylgnbu
bugn	gist_yarg	pastel2	rdpu	ylgn
bupu	gnbu	pink	rdylbu	ylorbr
cool	gray	piyg	rdylgn	ylorrd
copper	greens	prgn	reds	
dark2	greys	prism	set1	

改变颜色

```
import numpy as np
from mayavi import mlab

# 建立数据
x, y = np.mgrid[-10:10:200j, -10:10:200j]
z = 100 * np.sin(x * y) / (x * y)
# 对数据进行可视化
mlab.figure(bgcolor=(1, 1, 1))
surf = mlab.surf(z, colormap='cool')
# 更新视图并显示出来
mlab.show()
```

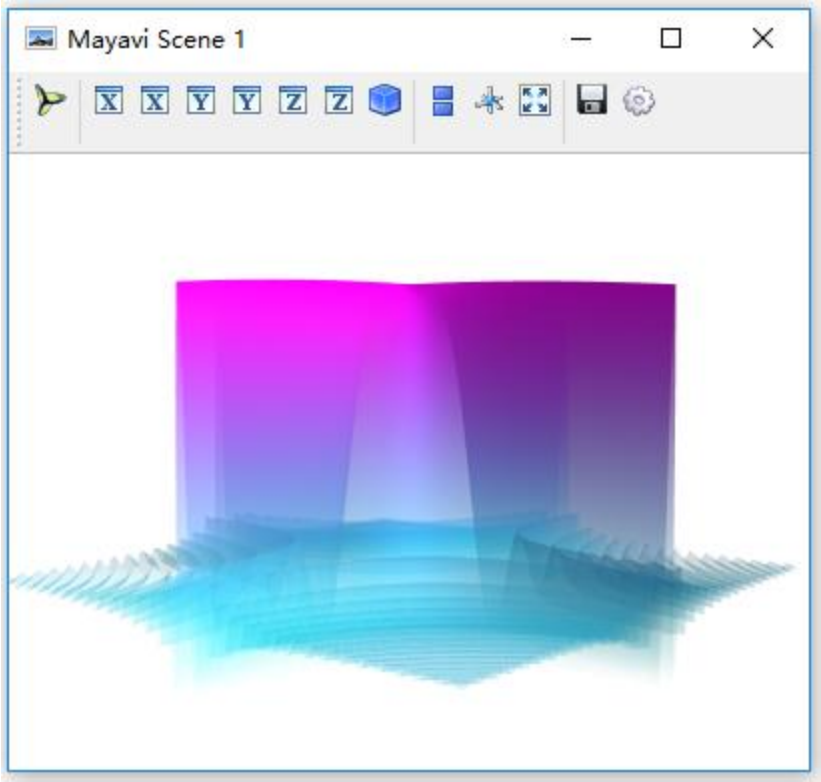
改变颜色



改变颜色

```
import numpy as np
from mayavi import mlab
#建立数据
x, y = np.mgrid[-10:10:200j, -10:10:200j]
z = 100 * np.sin(x * y) / (x * y)
# 对数据进行可视化
mlab.figure(bgcolor=(1, 1, 1))
surf = mlab.surf(z, colormap='cool')
# 访问surf对象的LUT
# LUT是一个255x4的数组，列向量表示RGBA，每个值得范围从0-255
lut = surf.module_manager.scalar_lut_manager.lut.table.to_array()
# 增加透明梯度，修改alpha通道
lut[:, -1] = np.linspace(0, 255, 256)
surf.module_manager.scalar_lut_manager.lut.table = lut
# 更新视图并显示出来
mlab.show()
```

改变颜色



mlab控制函数

图像控制函数

函数名称	说明
clf	清空当前图像 mlab.clf(figure=None)
close	关闭图像窗口 mlab.close(scene=None, all=False)
draw	重新绘制当前图像mlab.close(figure=None)
figure	建立一个新的Scene或者访问一个存在的Scene mlab.figure(figure=None,bgcolor=None,fgcolor=None, engine=None,size=(400,350))
gcf	返回当前图像的handle mlab.gcf(figure=None)
savefig	存储当前的前景，输出为一个文件，如png、jpg、bmp、tiff、 pdf、obj、vrml等

图像装饰函数

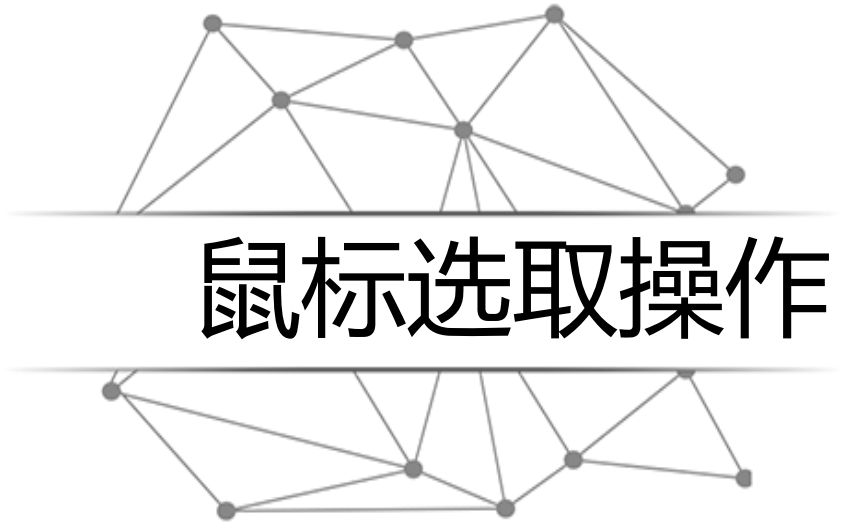
函数名称	说明
cololorbar	为对象的颜色映射增加颜色条 mlab.clolorbar(object=None, title=None, orientation=None, nb_labels=None, nb_colors=None, label_fmt=None)
scalarbar	为对象的标量颜色映射增加颜色条
vectorbar	为对象的矢量颜色映射增加颜色条
xlabel	建立坐标轴，并添加x轴的标签mlab.xlabel(text, object=None)
ylabel	建立坐标轴，并添加y轴的标签
zlabel	建立坐标轴，并添加z轴的标签

相机控制函数

函数名称	说明
move	移动相机和焦点 mlab.move(forward=None, right=None, up=None)
pitch	沿着“向右”轴旋转角度mlab.pitch(degrees)
roll	设置/获取相机沿“向前”轴旋转一定角度 mlab.roll(roll=None, figure=None)
view	设置/获取当前视图中相机的视点 mlab.view(azimuth=None, elevation=None, distance=None, focalpoint=None, roll=None, reset_roll=True, figure=None)
yaw	沿着“向上”轴旋转一定角度，mlab.yaw(degrees)

其他控制函数

函数名称	说明
animate	动画控制函数 mlab.animate(func=None, delay=500, ui=True)
axes	为当前物体设置坐标轴 mlab.axes(*args, **kwargs)
outline	为当前物体建立外轮廓 mlab.outline(*args, **kwargs)
show	与当前图像开始交互 mlab.show(func=None, stop=False)
show_pipeline	显示mayavi的管线对话框，可一进行场景属性的设置和编辑
text	为图像添加文本mlab.text(*args, **kwargs)
title	为绘制图像建立标题 mlab.title(*args, **kwargs)



鼠标选取操作

鼠标选取

- 选取一个物体，查看数据
- 选取物体上一点，查看数据

鼠标选取

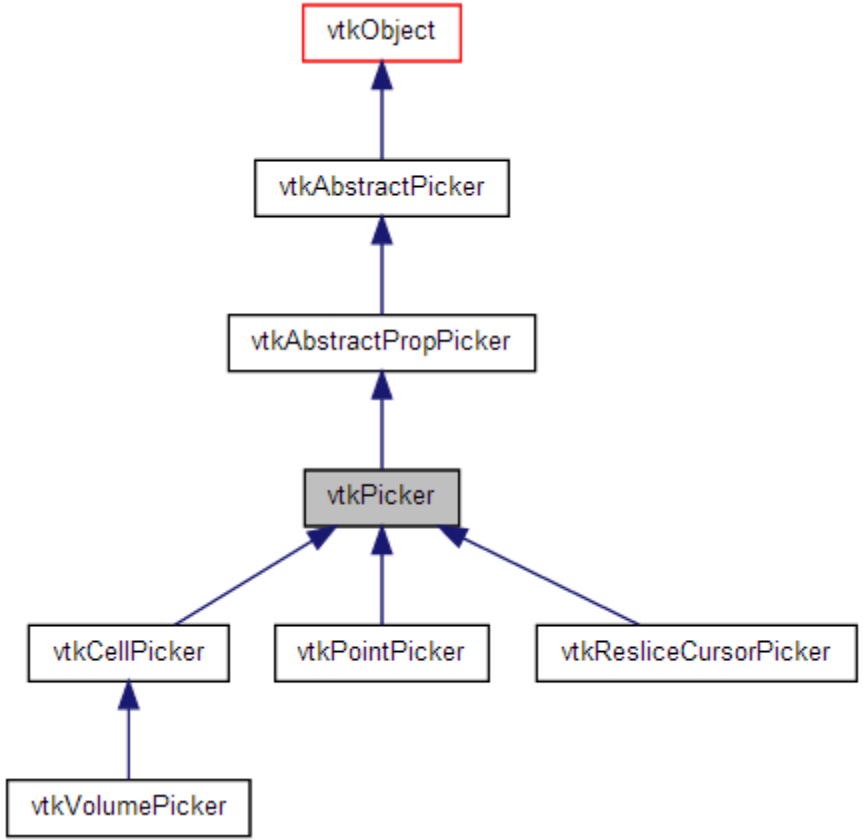
on_mouse_pick(callback, type='point',Button='Left',Remove=False)

Type:'point','cell'or 'world'

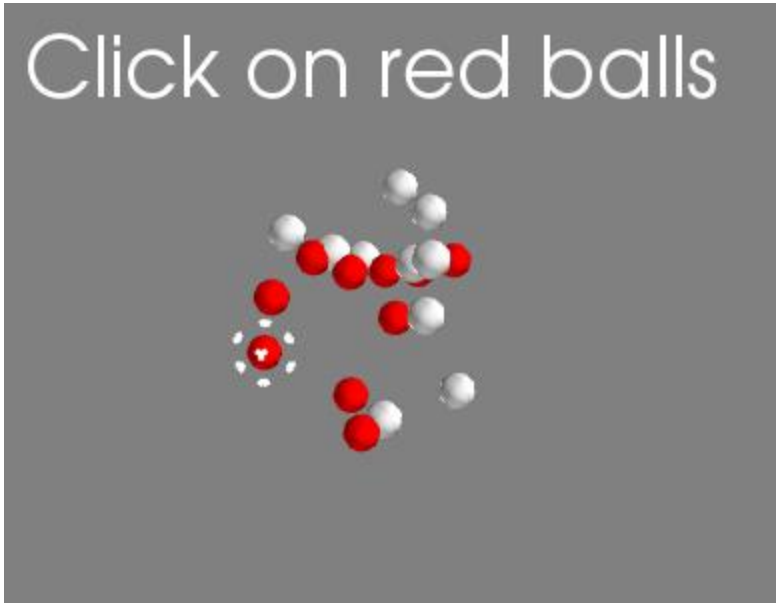
Button:'Left','Middle'or 'Right'

Remove:如果值为True，则callback函数不起作用

返回：一个vtk picker 对象


vtkObject
vtkAbstractPicker
vtkAbstractPropPicker
vtkPicker
vtkCellPicker
vtkPointPicker
vtkResliceCursorPicker
vtkVolumePicker

选取红色小球问题分析



- 建立一个figure
- 随机生成红、白小球
- 初始化红色小球选取外框
- 鼠标选取任意红色小球，外框移动到该小球上（callback）
- 建立on_mouse_pick()响应机制

程序框架

```
# 场景初始化
figure = mlab.gcf()
# 用mlab.points3d建立红色和白色小球的集合
#
… …
# 处理选取事件
def picker_callback(picker)
# 建立响应机制
picker = figure.on_mouse_pick(picker_callback)
mlab.show()
```

小球场景初始化建立

```
# 用mlab.points3d建立红色和白色小球的集合
x1, y1, z1 = np.random.random((3, 10))
red_glyphs = mlab.points3d(x1, y1, z1, color=(1, 0, 0),
                           resolution=10)
x2, y2, z2 = np.random.random((3, 10))
white_glyphs = mlab.points3d(x2, y2, z2, color=(0.9, 0.9, 0.9),
                             resolution=10)
```

选取框初始化建立

```
# 绘制选取框，并放在第一个小球上
outline = mlab.outline(line_width=3)
outline.outline_mode = 'cornered'
outline.bounds = (x1[0] - 0.1, x1[0] + 0.1,
                  y1[0] - 0.1, y1[0] + 0.1,
                  z1[0] - 0.1, z1[0] + 0.1)
```

“选取”回调函数的结构

```
#当选取事件发生时，调用此函数
def picker_callback(picker):
    if picker.actor in red_glyphs.actor.actors:
        # 计算哪个小球被选取
        # 确定该小球的ID
            # 找到与此红色小球相关的坐标
            # 将选取外框移到小球上
```

如何计算哪小球被选取

glyph_points：获取一个小球的顶点坐标列表

```
glyph_points =
red_glyphs.glyph.glyph_source.glyph_source.output.points.to_array()
```

如何计算哪小球被选取

```
def picker_callback(picker):
    if picker.actor in red_glyphs.actor.actors:
        # 计算被选取的小球的ID号
        point_id = int(picker.point_id / glyph_points.shape[0])
```

Picker对像选取的顶点ID　　每一个小球顶点的总数

如何计算哪小球被选取

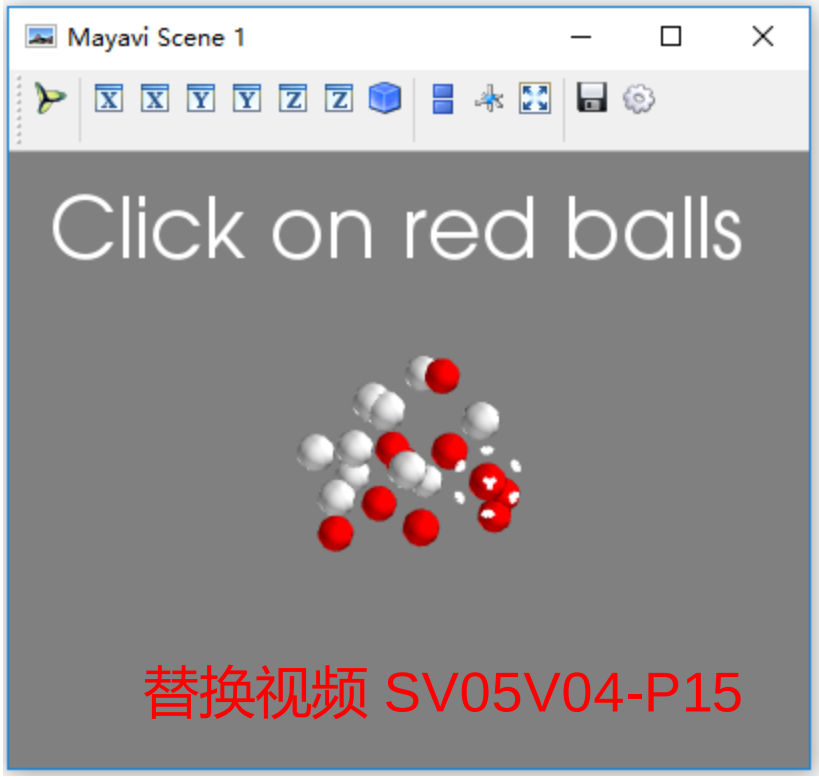
```
def picker_callback(picker):
    if picker.actor in red_glyphs.actor.actors:
        # 计算哪个小球被选取
        point_id = int(picker.point_id / glyph_points.shape[0])
        if point_id != -1:
            # 计算与此红色小球相关的坐标
            x, y, z = x1[point_id], y1[point_id], z1[point_id]
            # 将外框移到小球上
            outline.bounds = (x - 0.1, x + 0.1,
                              y - 0.1, y + 0.1,
                              z - 0.1, z + 0.1)
```

建立响应机制

```
picker = figure.on_mouse_pick(picker_callback)
mlab.title(‘Click on red balls’)#设置窗口的标题文字
mlab.show()
```

```
from mayavi import mlab

######场景初始化######
figure = mlab.gcf()

# 用mlab.points3d建立红色和白色小球的集合
x1, y1, z1 = np.random.random((3, 10))
red_glyphs = mlab.points3d(x1, y1, z1, color=(1, 0, 0),
                           resolution=10)
x2, y2, z2 = np.random.random((3, 10))
white_glyphs = mlab.points3d(x2, y2, z2, color=(0.9, 0.9, 0.9),
                             resolution=10)

# 绘制选取框，并放在第一个小球上
outline = mlab.outline(line_width=3)
outline.outline_mode = 'cornered'
outline.bounds = (x1[0] - 0.1, x1[0] + 0.1,
                  y1[0] - 0.1, y1[0] + 0.1,
                  z1[0] - 0.1, z1[0] + 0.1)

######处理选取事件#####
# 获取构成一个红色小球的顶点列表
glyph_points = red_glyphs.glyph.glyph_source.glyph_source.output.points.to_array()
#当选取事件发生时调用此函数
def picker_callback(picker):
    if picker.actor in red_glyphs.actor.actors:
        # 计算哪个小球被选取
        point_id = int(picker.point_id / glyph_points.shape[0])  # int向下取整
        if point_id != -1:#如果没有小球被选取，则point_id = -1
            # 找到与此红色小球相关的坐标
            x, y, z = x1[point_id], y1[point_id], z1[point_id]
            # 将外框移到小球上
            outline.bounds = (x - 0.1, x + 0.1,
                              y - 0.1, y + 0.1,
                              z - 0.1, z + 0.1)

picker = figure.on_mouse_pick(picker_callback)
mlab.title('Click on red balls')
mlab.show()
```

程序的运行结果



程序框架的优化

程序运行两个问题：

- 小球初始速度太慢
- 鼠标选取不精确

程序框架的优化

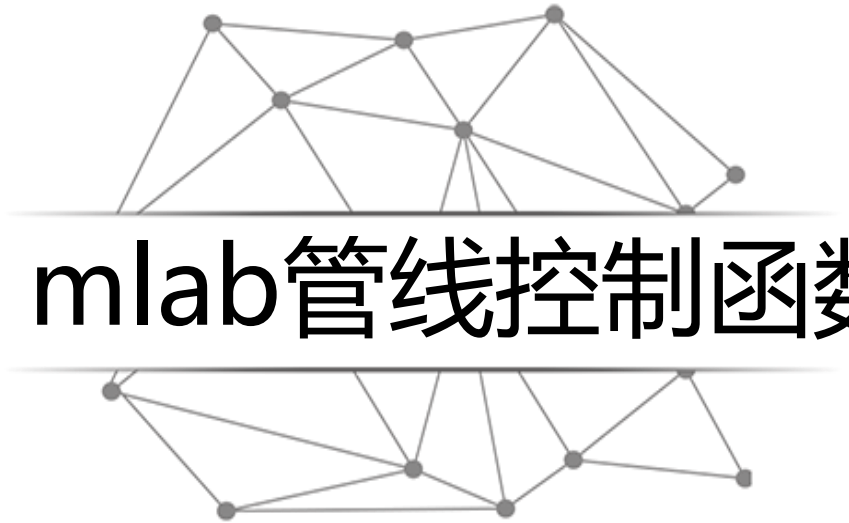
```
# 场景初始化
figure = mlab.gcf()
figure.scene.disable_render = True
# 用mlab.points3d建立红色和白色小球的集合
… …
figure.scene.disable_render = False
# 处理选取事件
def picker_callback(picker)
# 建立响应机制
picker = figure.on_mouse_pick(picker_callback)
mlab.show()
```

所有物体全部建立完再绘制！

程序框架的优化

```
# 场景初始化
figure = mlab.gcf()
figure.scene.disable_render = True
# 用mlab.points3d建立红色和白色小球的集合
… …
figure.scene.disable_render = False
# 处理选取事件
def picker_callback(picker)
# 建立响应机制
picker = figure.on_mouse_pick(picker_callback)
Picker.tolerance = 0.01
mlab.show()
```

设置tolerance参数，提高选取精度！



mlab管线控制函数

Mlab管线控制函数的调用

Sources：数据源
Filters：用来数据变换
Modules：用来实现可视化

```
mlab.pipeline.function()
```

Sources

函数名称	说明
grid_source	建立二维网格数据
line_source	建立线数据
open	打开一个数据文件
scalar_field	建立标量场数据
vector_field	建立矢量场数据
volume_filed	建立体数据

Filters

Filters	说明
contour	对输入数据集计算等值面
cut_plane	对数据进行切面计算，可以交互的更改和移动切面
delaunay2D	执行二维delaunay三角化
delaunay3D	执行三维delaunay三角化

Filters

Filters	说明
extract_grid	允许用户选择structured grid的一部分数据
extract_vector_norm	计算数据矢量的法向量，特别用于在计算矢量数据的梯度时
mask_points	对输入数据进行采样
threshold	取一定阈值范围内的数据
transform_data	对输入数据执行线性变换
tube	将线转成管线数据

Modules

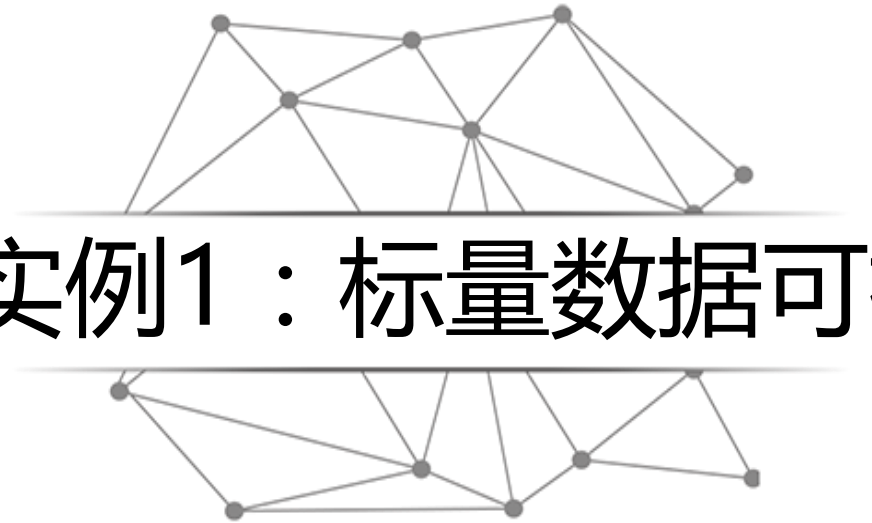
Modules	说明
axes	绘制坐标轴
glyph	对输入点绘制不同类型的符号，符号的颜色和方向由该点的标量和适量数据决定。
image_plane_widget	绘制某一平面数据的细节
iso_surface	对输入的体数据绘制其等值面

Modules

Modules	说明
outline	对输入数据绘制外轮廓
scalar_cut_plane	对输入的标量数据绘制特定位置的切平面
streamline	对矢量数据绘制流线
surface	对数据（VTK dataset，mayavi sources）建立外表面
text	绘制一段文本
vector_cut_plane	对输入的矢量数据绘制特定位置的切平面
volume	对标量场数据进行体绘制

Mlab Reference

http://docs.enthought.com/mayavi/mayavi/auto/mlab_reference.html



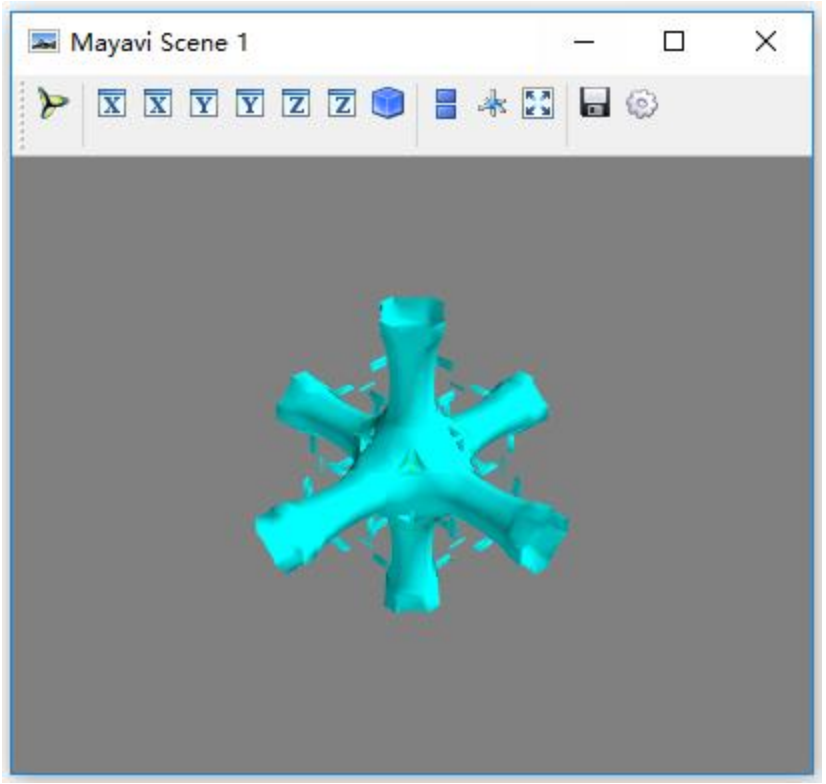
实例1：标量数据可视化

生成标量数据

```
import numpy as np
x, y, z = np.ogrid[-10:10:20j, -10:10:20j, -10:10:20j]
s = np.sin(x*y*z)/(x*y*z)

from mayavi import mlab
mlab.contour3d(s)
mlab.show()
```

等值面绘制

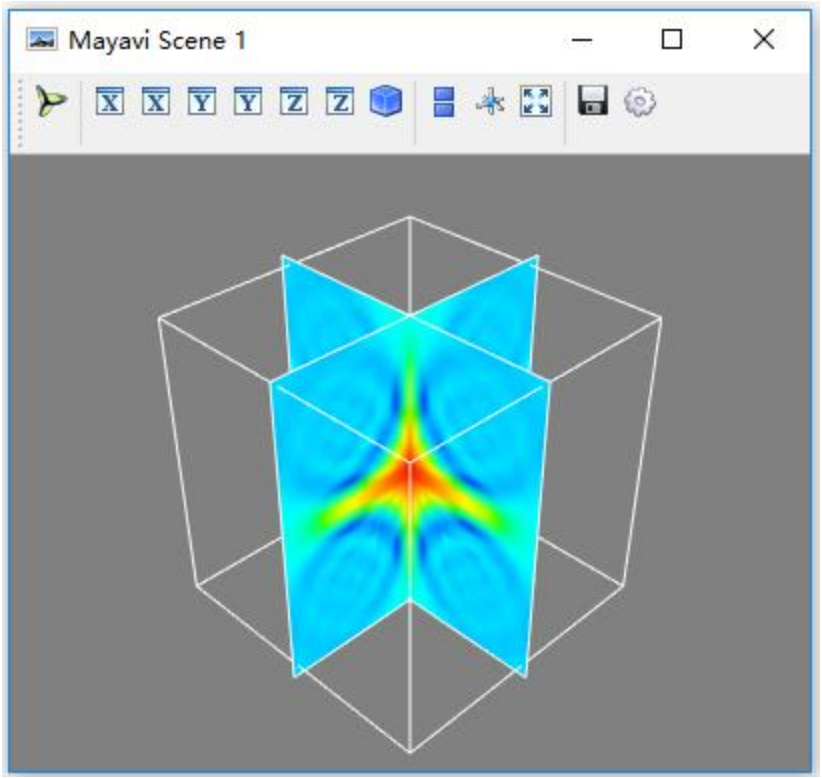
```
mlab.contour3d(s)
```



切平面

```
from mayavi import mlab
from mayavi.tools import pipeline
mlab.pipeline.image_plane_widget(mlab.pipeline.scalar_field(s),
                                 plane_orientation='x_axes',
                                 slice_index=10,
                            )
mlab.pipeline.image_plane_widget(mlab.pipeline.scalar_field(s),
                                 plane_orientation='y_axes',
                                 slice_index=10,
                            )
mlab.outline()
```

切平面



复合观测方法

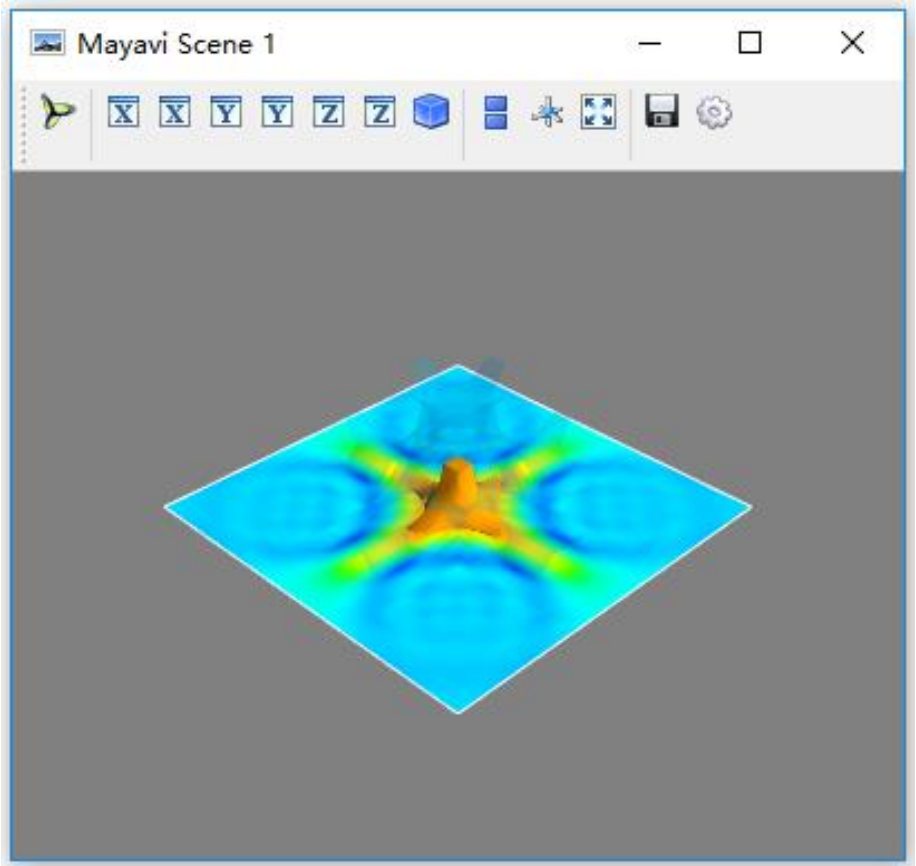
```
from mayavi import mlab
from mayavi.tools import pipeline
src = mlab.pipeline.scalar_field(s)
mlab.pipeline.iso_surface(src, contours=[s.min()+0.1*s.ptp(), ], opacity=0.1)
mlab.pipeline.iso_surface(src, contours=[s.max()-0.1*s.ptp(), ])
mlab.pipeline.image_plane_widget(src,
                                 plane_orientation='z_axes',
                                 slice_index=10,
                            )
```

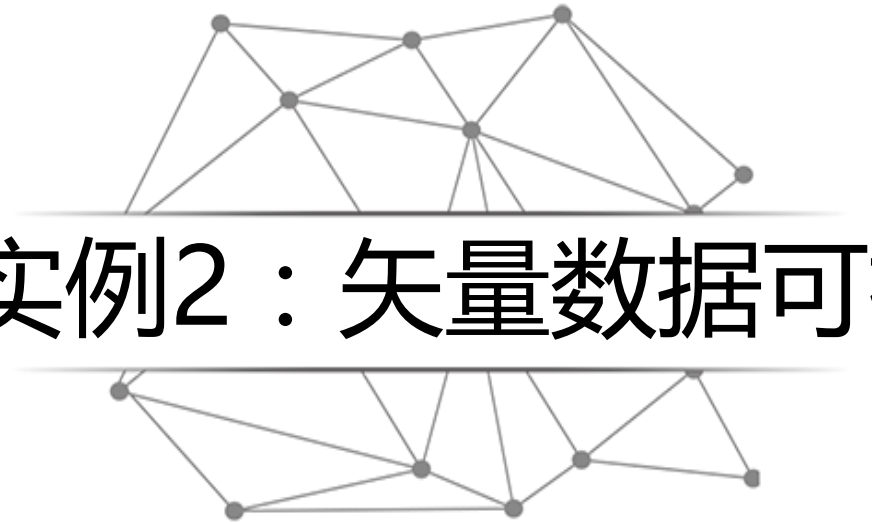
复合观测方法

```
mlab.pipeline.iso_surface(src,
contours=[s.min()+0.1*s.ptp(), ], opacity=0.1)
```

```
mlab.pipeline.iso_surface(src,
contours=[mlab.pipeline.scalar_cut_plane(src), ])
```

复合观测方法





实例2：矢量数据可视化

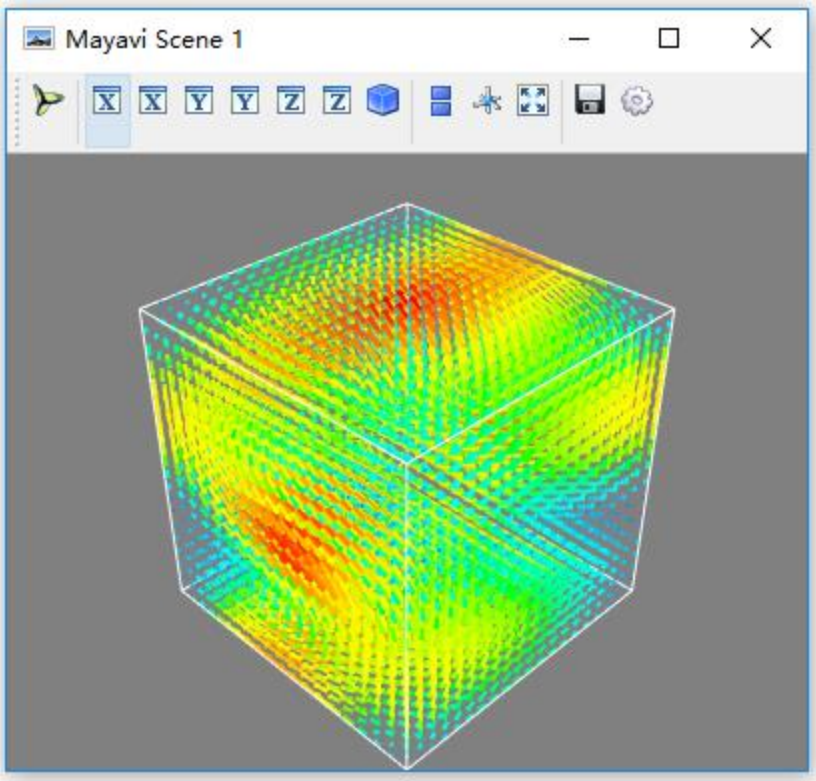
生成矢量数据

```
import numpy as np
x, y, z = np.mgrid[0:1:20j, 0:1:20j, 0:1:20j]

u =    np.sin(np.pi*x) * np.cos(np.pi*z)
v = -2*np.sin(np.pi*y) * np.cos(2*np.pi*z)
w = np.cos(np.pi*x)*np.sin(np.pi*z) + np.cos(np.pi*y)*np.sin(2*np.pi*z)
```

Quiver绘制

```
from mayavi import mlab
mlab.quiver3d(u,v,w)
mlab.outline()

mlab.show()
```

Quiver绘制



Masking Vector采样

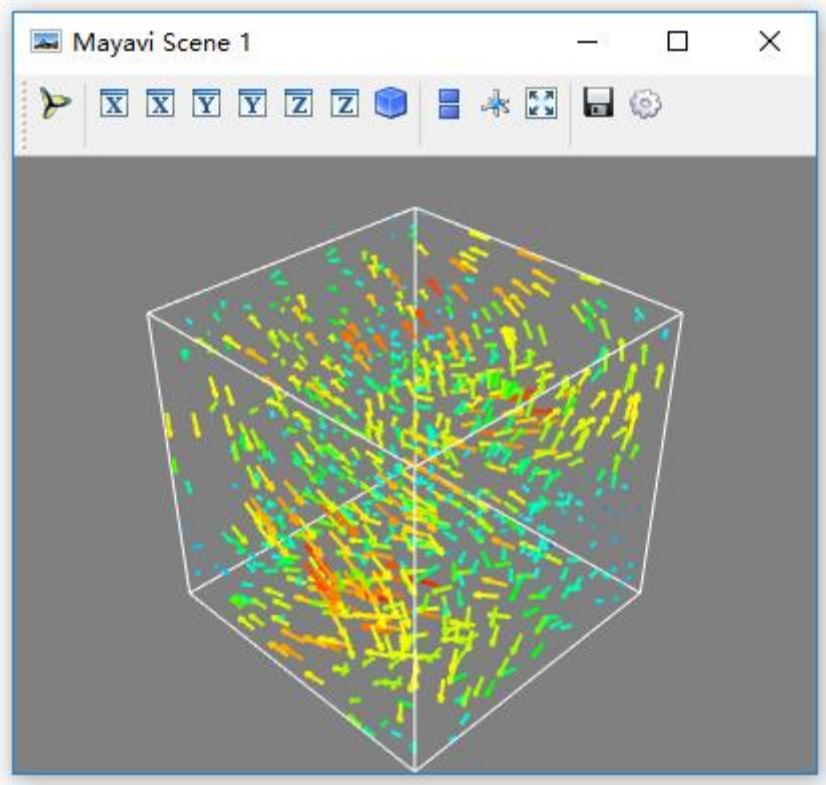
```
from mayavi import mlab
src = mlab.pipeline.vector_field(u, v, w)
mlab.pipeline.vectors(src, mask_points=10, scale_factor=2.0)

mlab.show()
```

Masking Vector采样

可尝试：

```
from mayavi import mlab
vectors = mlab.quiver3d(u,v,w)
vectors.glyph.mask_input_points = True
vectors.glyph.mask_points.on_ratio =10
vectors.glyph.glyph.scale_factor = 2.0

#src = mlab.pipeline.vector_field(u, v, w)
#mlab.pipeline.vectors(src, mask_points=10, scale_factor=2.0)
#mlab.quiver3d(u,v,w)
mlab.outline()
mlab.show()
```

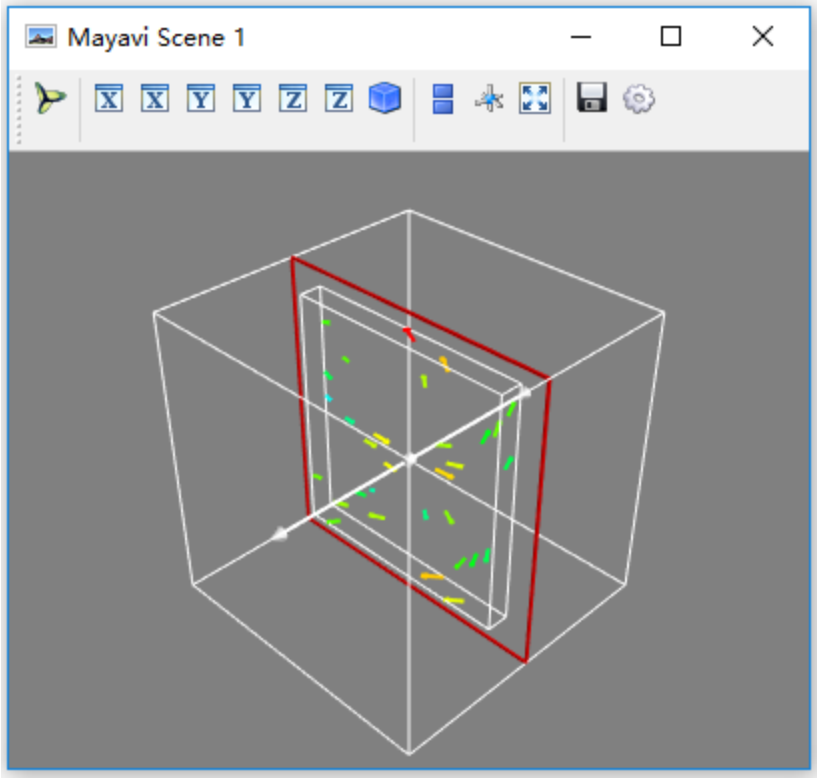
Masking Vector采样



Cut Plane切面

```
from mayavi import mlab
src = mlab.pipeline.vector_field(u, v, w)
mlab.pipeline.vector_cut_plane(src, mask_points=10, scale_factor=2)

mlab.show()
```

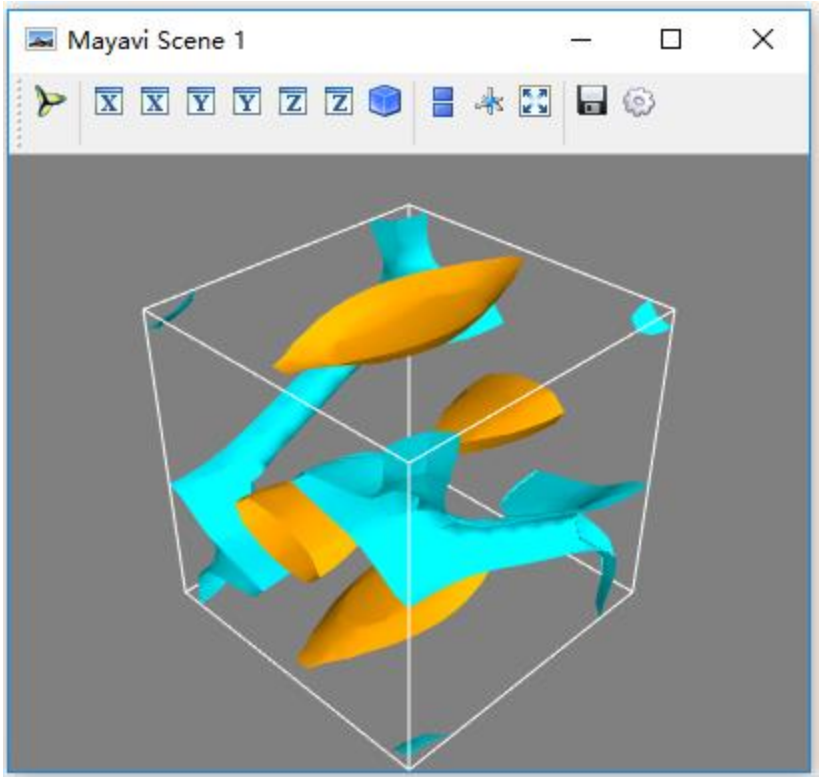
Cut Plane切面



级数的等值面

```
from mayavi import mlab
src = mlab.pipeline.vector_field(u, v, w)
magnitude = mlab.pipeline.extract_vector_norm(src)
mlab.pipeline.iso_surface(magnitude, contours=[2.0, 0.5])
mlab.outline()

mlab.show()
```

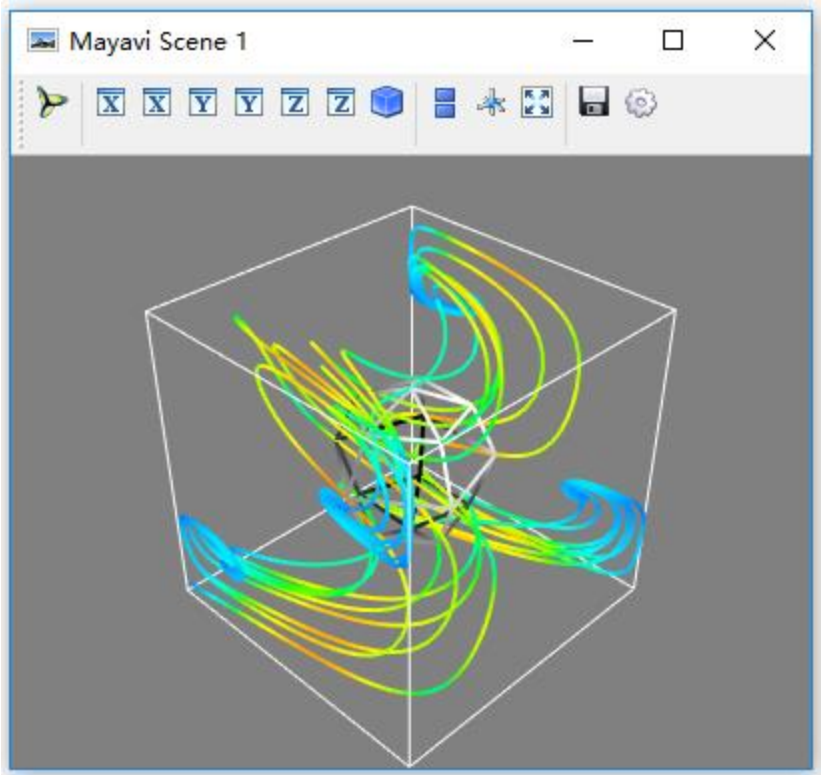
级数的等值面



Flow可视化

```
from mayavi import mlab
flow = mlab.flow(u, v, w, seed_scale=1,
                          seed_resolution=5,
                          integration_direction='both')
mlab.outline()

mlab.show()
```

级数的等值面



复合观测方法

```
iso = mlab.pipeline.iso_surface(magnitude, contours=[2.0, ], opacity=0.3)
vec = mlab.pipeline.vectors(magnitude, mask_points=40,
                                       line_width=1,
                                       color=(0.8, 0.8, 0.8),
                                       scale_factor=4.)
flow = mlab.pipeline.streamline(magnitude, seedtype='plane',
                                           seed_visible=False,
                                           seed_scale=0.5,
                                           seed_resolution=1,
                                           linetype='ribbon',)
vcp = mlab.pipeline.vector_cut_plane(magnitude, mask_points=2,
                                           scale_factor=4,
                                           colormap='jet',
                                           plane_orientation='x_axes')
```

符合观测方法