第三次上机作业

**18029100040**

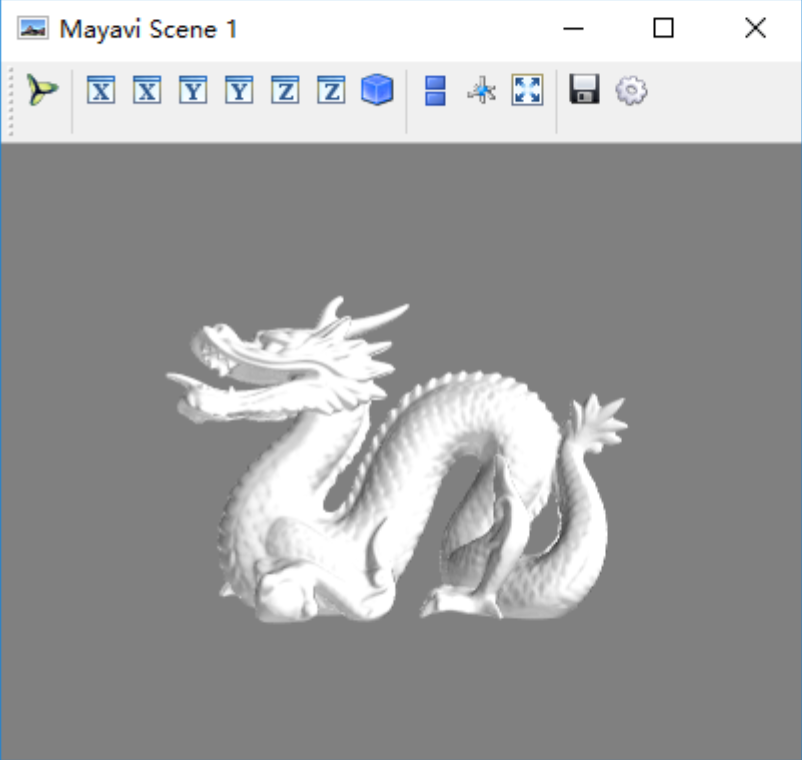
**吴程锴**

# 作业 19：Dragon 绘制：

## 代码

1. **from** mayavi **import** mlab
2. **from** os.path **import** join
3. **import** tarfile
5. # 读取tar压缩文件
6. dragon\_tar\_file = tarfile.open('dragon.tar.gz')
7. **try**:
8. os.mkdir('dragon\_data')
9. **except**:
10. **pass**
11. dragon\_tar\_file.extractall('dragon\_data')
12. dragon\_tar\_file.close()
13. dragon\_ply\_file = join('dragon\_data', 'dragon\_recon', 'dragon\_vrip.ply')
15. # 渲染dragon ply文件
16. mlab.pipeline.surface(mlab.pipeline.open(dragon\_ply\_file))
17. mlab.show()
19. # 删除解压的文件夹
20. **import** shutil
22. shutil.rmtree('dragon\_data')

## 结果

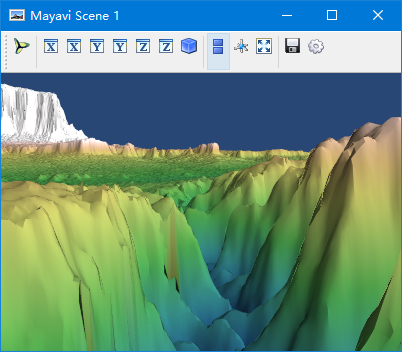


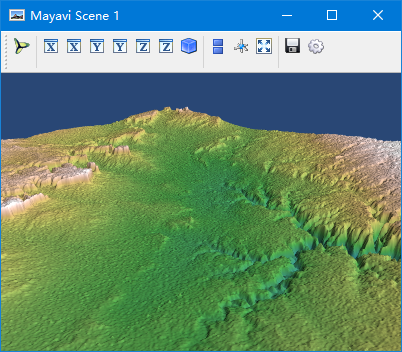
# 作业 20：地形可视化

## 代码

1. **import** zipfile
2. **import** numpy as np
3. **from** mayavi **import** mlab
5. # 读取压缩文件
6. hgt = zipfile.ZipFile('5.1.6 N36W113.hgt.zip').read('N36W113.hgt')
7. data = np.fromstring(hgt, '>i2')
8. data.shape = (3601, 3601)
9. data = data.astype(np.float32)
10. data = data[:1000, 900:1900]
11. data[data == -32768] = data[data > 0].min()
13. # 渲染地形hgt的数据data
14. mlab.figure(size=(400, 320), bgcolor=(0.16, 0.28, 0.46))
15. mlab.surf(data, colormap='gist\_earth', warp\_scale=0.2,
16. vmin=1200, vmax=1610)
18. # 清空内存
19. **del** data
20. # 创建交互式的可视化窗口
21. mlab.view(-5.9, 83, 570, [5.3, 20, 238])
22. mlab.show()

## 结果



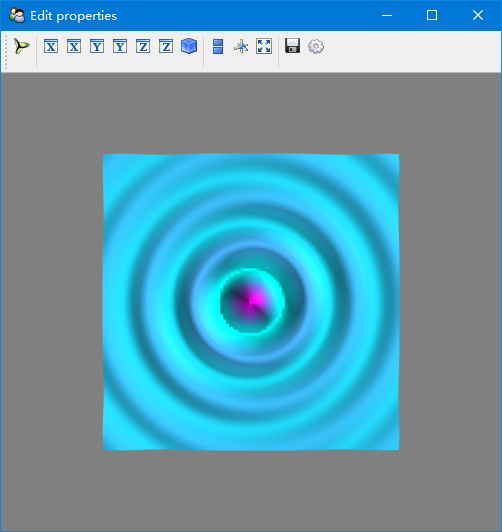


# 作业 21：建立简单的mayavi窗口

## 代码

1. **from** numpy **import** sqrt, sin, mgrid
2. **from** traits.api **import** HasTraits, Instance
3. **from** traitsui.api **import** View, Item
4. **from** tvtk.pyface.scene\_editor **import** SceneEditor
5. **from** mayavi.tools.mlab\_scene\_model **import** MlabSceneModel
6. **from** mayavi.core.ui.mayavi\_scene **import** MayaviScene
8. **class** ActorViewer(HasTraits):
9. # 场景模型
10. scene = Instance(MlabSceneModel, ())
11. # 建立视图
12. view = View(Item(name='scene',
13. editor=SceneEditor(scene\_class=MayaviScene),
14. show\_label=False,
15. resizable=True,
16. height=500,
17. width=500),
18. resizable=True)
20. **def** \_\_init\_\_(self, \*\*traits):
21. HasTraits.\_\_init\_\_(self, \*\*traits)
22. self.generate\_data()
24. **def** generate\_data(self):
25. # 建立数据
26. X, Y = mgrid[-2:2:100j, -2:2:100j]
27. R = 10\*sqrt(X\*\*2 + Y\*\*2)
28. Z = sin(R)/R
29. # 绘制数据
30. self.scene.mlab.surf(X, Y, Z, colormap='cool')
32. a = ActorViewer()
33. a.configure\_traits()

## 结果

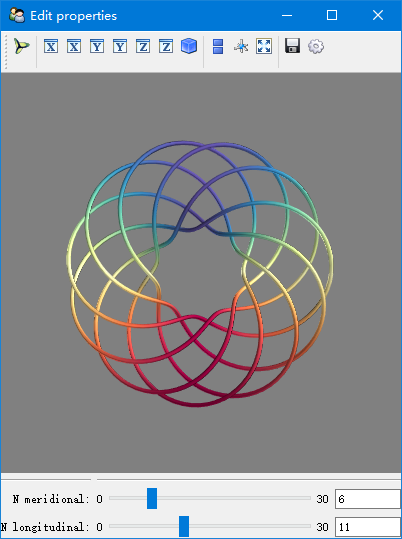


# 作业 22：基于交互控制的Mayavi窗口

## 代码

1. **from** traits.api **import** HasTraits, Range, Instance, on\_trait\_change
2. **from** traitsui.api **import** View, Item, Group
3. **from** mayavi.core.api **import** PipelineBase
4. **from** mayavi.core.ui.api **import** MayaviScene, SceneEditor, MlabSceneModel
6. **from** numpy **import** arange, pi, cos, sin
7. dphi = pi/300.
8. phi = arange(0.0, 2\*pi + 0.5\*dphi, dphi, 'd')
9. **def** curve(n\_mer, n\_long):
10. mu = phi\*n\_mer
11. x = cos(mu) \* (1 + cos(n\_long \* mu/n\_mer)\*0.5)
12. y = sin(mu) \* (1 + cos(n\_long \* mu/n\_mer)\*0.5)
13. z = 0.5 \* sin(n\_long\*mu/n\_mer)
14. t = sin(mu)
15. **return** x, y, z, t
17. **class** MyModel(HasTraits):
18. n\_meridional    = Range(0, 30, 6)
19. n\_longitudinal  = Range(0, 30, 11)
20. # 场景模型实例
21. scene = Instance(MlabSceneModel, ())
22. # 管线实例
23. plot = Instance(PipelineBase)
24. #当场景被激活，或者参数发生改变，更新图形
25. @on\_trait\_change('n\_meridional,n\_longitudinal,scene.activated')
26. **def** update\_plot(self):
27. x, y, z, t = curve(self.n\_meridional, self.n\_longitudinal)
28. **if** self.plot **is** None:#如果plot未绘制则生成plot3d
29. self.plot = self.scene.mlab.plot3d(x, y, z, t,
30. tube\_radius=0.025, colormap='Spectral')
31. **else**:#如果数据有变化，将数据更新即重新赋值
32. self.plot.mlab\_source.set(x=x, y=y, z=z, scalars=t)
34. # 建立视图布局
35. view = View(Item('scene', editor=SceneEditor(scene\_class=MayaviScene),
36. height=250, width=300, show\_label=False),
37. Group('\_', 'n\_meridional', 'n\_longitudinal'),
38. resizable=True)
40. model = MyModel()
41. model.configure\_traits()

## 结果

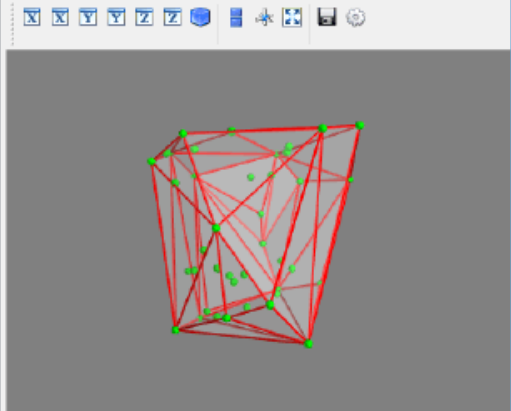


# 作业 23：Spatial-ConvexHull三维凸包可视化

## 代码

1. **from** tvtk.api **import** tvtk
2. **import** numpy as np
3. **from** scipy.spatial **import** ConvexHull
4. **import** matplotlib.pyplot as plt
5. **def** convexhull(ch3d): #1 定义凸多面体 tvtk 的 Polydata() 对象
6. poly = tvtk.PolyData()
7. poly.points = ch3d.points
8. poly.polys = ch3d.simplices
9. #2 定义凸多面体顶点的小球
10. sphere = tvtk.SphereSource(radius = 0.02)
11. points3d = tvtk.Glyph3D()
12. points3d.set\_source\_connection(sphere.output\_port)
13. points3d.set\_input\_data(poly)
14. #3 绘制凸多面体的面，设置半透明度
15. m1 = tvtk.PolyDataMapper()
16. m1.set\_input\_data(poly)
17. a1 = tvtk.Actor(mapper=m1)
18. a1.property.opacity = 0.3
19. #4 绘制凸多面体的边，设置为红色
20. m2 = tvtk.PolyDataMapper()
21. m2.set\_input\_data(poly)
22. a2 = tvtk.Actor(mapper=m2)
23. a2.property.representation = 'wireframe'
24. a2.property.line\_width = 2.0
25. a2.property.color = (1.0,0,0)
26. #5 绘制凸多面体的顶点，设置为绿色
27. m3 = tvtk.PolyDataMapper(input\_connection=points3d.output\_port)
28. a3 = tvtk.Actor(mapper = m3)
29. a3.property.color = (0.0,1.0,0.0)
30. **return** [a1,a2,a3]
31. #4. 定义绘制场景用的函数
32. **def** ivtk\_scene(actors):
33. **from** tvtk.tools **import** ivtk
34. win = ivtk.IVTKWithCrustAndBrowser() #创建 crust 窗口
35. win.open()
36. win.scene.add\_actor(actors)
37. dialog = win.control.centralWidget().widget(0).widget(0) #窗口错误修正
38. **from** pyface.qt **import** QtCore
39. dialog.setWindowFlags(QtCore.Qt.WindowFlags(0x00000000))
40. dialog.show()
41. **return** win
43. np.random.seed(42)
44. points3d=np.random.rand(40,3)
45. ch3d=ConvexHull(points3d)
46. actors=convexhull(ch3d)
47. win=ivtk\_scene(actors)

## 结果



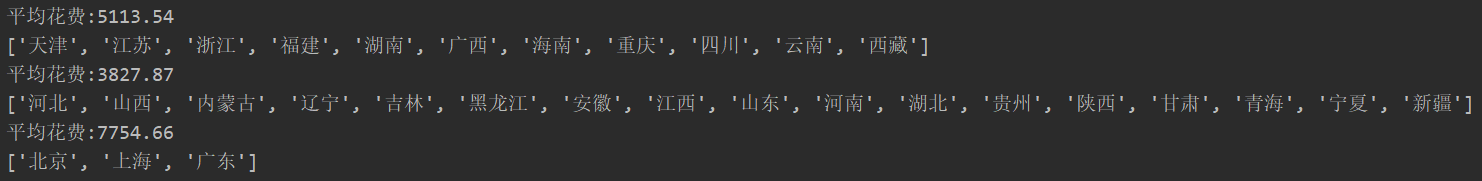
# 作业 24：K-means聚类

## 代码

1. **import** numpy as np
2. **from** sklearn.cluster **import** KMeans
3. #定义数据导入方法
4. **def** loadData(filePath):
5. fr=open(filePath,'r+')  #读写打开一个文本文件
6. lines=fr.readlines()    #一次读取整 个文件（类似于.read())
7. retData=[]              #城市各项信息
8. retCityName=[]          #城市名称
9. **for** line **in** lines:
10. items=line.strip().split(",")
11. retCityName.append(items[0])
12. retData.append([items[i] **for** i **in** range(1,len(items))])
13. #print(retCityName)
14. **return** retData,retCityName

17. #加载数据，创建K-means算法实例，并进行训练，获得标签
18. **if** \_\_name\_\_=='\_\_main\_\_':
19. filepath='city.txt'
20. data,cityName=loadData(filepath)  #利用loadData方法读取数据
21. km=KMeans(n\_clusters=3)              #创建实例
22. label=km.fit\_predict(data)          #调用Kmeans() fit\_predict()方法进行计算
23. expenses=np.sum(km.cluster\_centers\_,axis=1)
24. #print(expenses)
25. CityCluster=[[],[],[]]      #将城市 按label分成设定的簇，将每个簇的城市输出
26. **for** i **in** range(len(cityName)):
27. CityCluster[label[i]].append(cityName[i])
28. **for** i **in** range(len(CityCluster)):
29. **print**("Expenses:%.2f" %expenses[i])
30. **print**(CityCluster[i])

## 结果

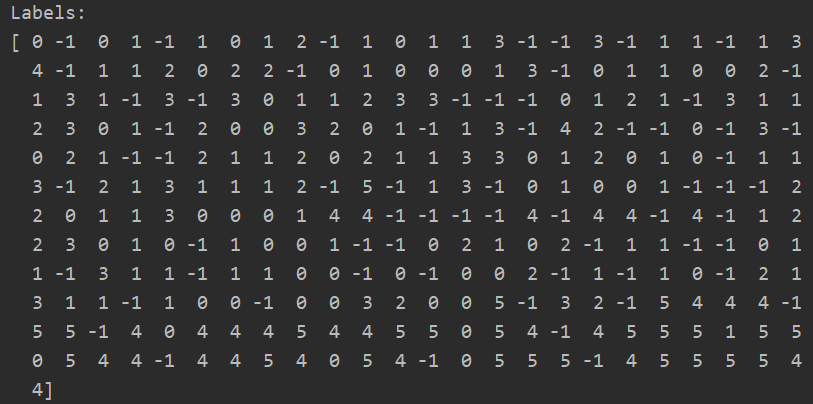


# 作业 25：DBSCAN聚类

## 代码

1. **import** numpy as np
2. **import** sklearn.cluster as skc
3. **from** sklearn **import** metrics
4. **import** matplotlib.pyplot as plt
6. mac2id = dict()
7. onlinetimes = []
8. f = open(r'学生月上网时间分布-TestData.txt', encoding='utf-8')
9. **for** line **in** f:
10. # 读取每条数据中的mac地址，开始上网时间，上网时长
11. mac = line.split(',')[2]
12. onlinetime = int(line.split(',')[6])
13. starttime = int(line.split(',')[4].split(' ')[1].split(':')[0])
15. # mac2id是一个字典：key是mac地址，value是对应mac地址的上网时长以及开始上网时间
16. **if** mac **not** **in** mac2id:
17. mac2id[mac] = len(onlinetimes)
18. onlinetimes.append((starttime, onlinetime))
19. **else**:
20. onlinetimes[mac2id[mac]] = [(starttime, onlinetime)]
21. real\_X = np.array(onlinetimes).reshape((-1, 2))
23. # 调用DBSCAN方法训练，labels为每个数据的簇标签
24. X = real\_X[:, 0:1]
25. db = skc.DBSCAN(eps=0.01, min\_samples=20).fit(X)
26. labels = db.labels\_
28. # 打印数据被记上的标签，计算标签为-1,即噪声数据的比例
29. **print**('Labels:')
30. **print**(labels)
31. raito = len(labels[labels[:] == -1]) / len(labels)
32. **print**('Noise raito:', format(raito, '.2%'))
34. # 计算簇的个数并打印，评价聚类效果
35. n\_clusters\_ = len(set(labels)) - (1 **if** -1 **in** labels **else** 0)
37. **print**('Estimated number of clusters: %d' % n\_clusters\_)
38. **print**("Silhouette Coefficient: %0.3f" % metrics.silhouette\_score(X, labels))
40. # 打印各簇标号以及各簇内数据
41. **for** i **in** range(n\_clusters\_):
42. **print**('Cluster ', i, ':')
43. **print**(list(X[labels == i].flatten()))
45. plt.hist(X, 24)
46. plt.show()

## 结果



1. Noise raito: 22.15%
2. Estimated number of clusters: 6
3. Silhouette Coefficient: 0.710
4. Cluster  0 :
5. [22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22]
6. Cluster  1 :
7. [23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23]
8. Cluster  2 :
9. [20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20]
10. Cluster  3 :
11. [21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21]
12. Cluster  4 :
13. [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
14. Cluster  5 :
15. [7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7]