



# 城市空间建模与仿真

第八讲 城市空间三维数据表达和重建-3D数据配准与表面重建

任课教师：汤圣君  
建筑与城市规划学院 城市空间信息工程系



# 目录

## CONTENTS

- 01 3D数据配准**
- 02 3D表面重建**
- 03 课堂练习**



# 目录

## CONTENTS

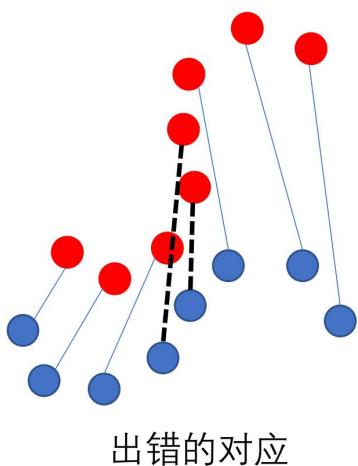
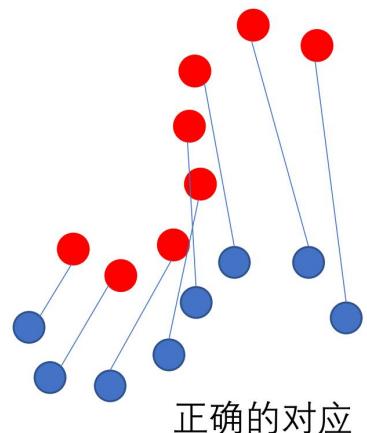
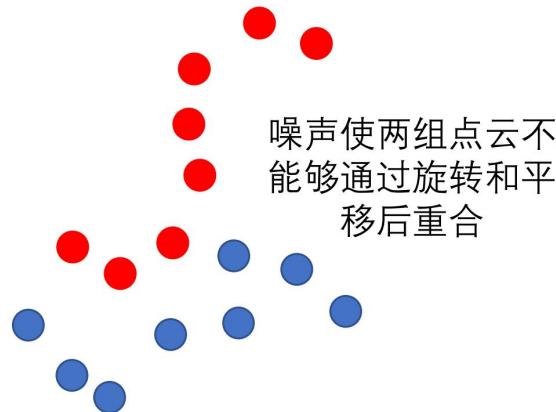
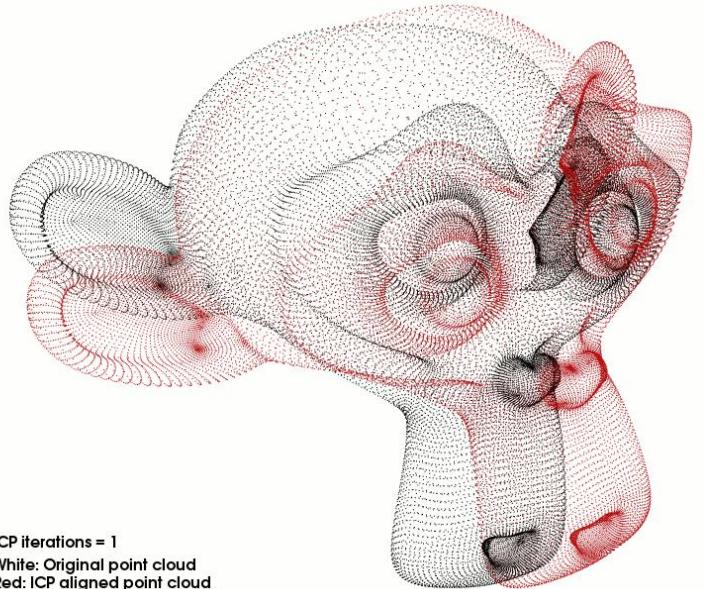
01

3D数据配准

# 3D数据配准



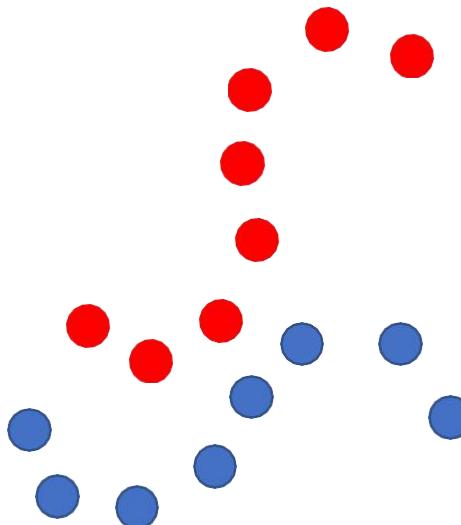
- 具体应用——多个视角获得的点云融合，得到完整点云模型
- 目的：找到旋转平移矩阵，使得两组点云重合
- 问题的难点（见下面的例子）
  - 噪声和干扰
  - 点云对应关系错误
  - 两组点云中各存在一部分点对应着不重合的物体表面



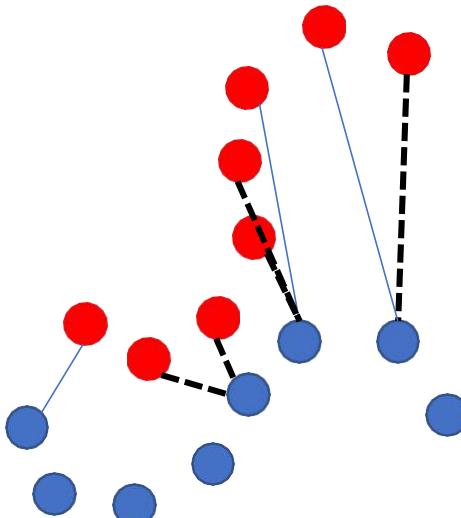
# 3D数据配准



如何解决点云对应关系错误以及噪声和干扰?  
用最近邻算法



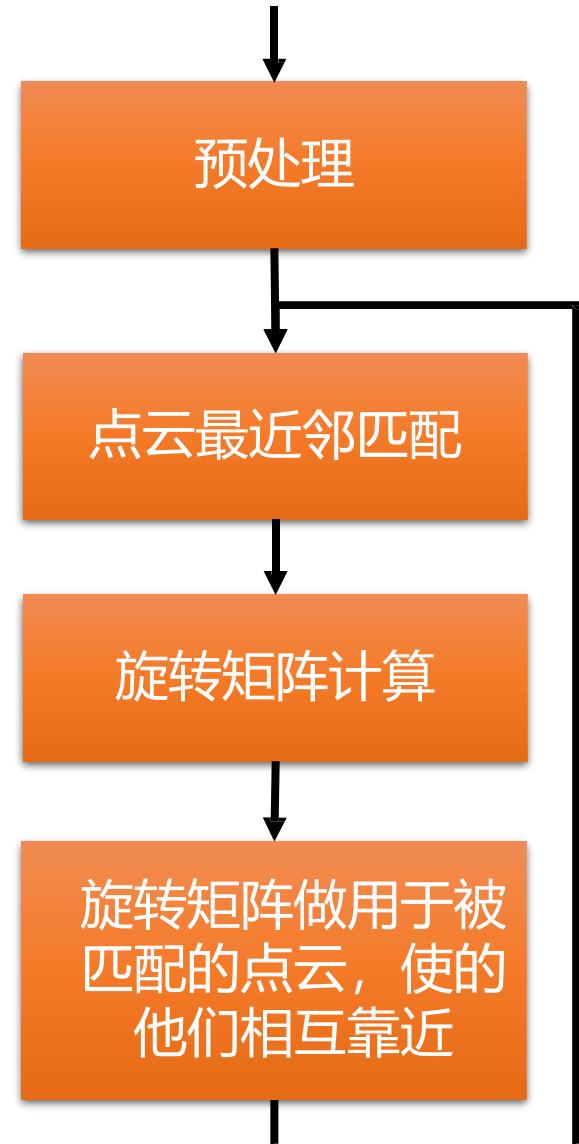
对每个红点，找到对应最近距离蓝点



⌚错了不少  
😊只要错的不是很大，  
还是能够进行匹配 比  
如：错误点相对较少，  
或者错误的匹配点和  
正确的匹配点距离很近

最近邻算法不一定找到最优化配准点?  
使用迭代算法（见下一页）

## 算法流程

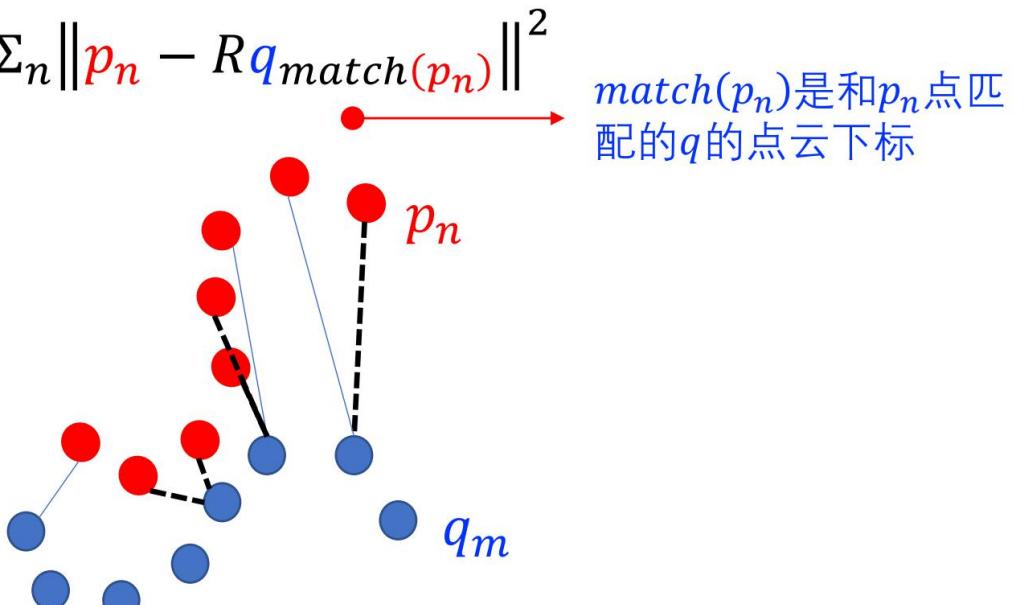


- 通过“中心化”实现平移量对齐
- (两组点云分别扣除他们的平均坐标)
- 经过预处理后，后面只考虑旋转变换
- 这一迭代规程不断重复，直到两组点云匹配距离足够小

# 3D数据配准



1. 预处理——点云平移:  $\begin{cases} p_n \leftarrow p_n - \bar{p} \\ q_n \leftarrow q_n - \bar{q} \end{cases}$ , 其中  $\begin{cases} \bar{p} = \frac{1}{N} \sum_n p_n \\ \bar{q} = \frac{1}{N} \sum_n q_n \end{cases}$  是点云中心(平均坐标)
2. 点云匹配——调整一个点云的下标, 使得两个点云相同下标对应的点是匹配的  
(一般就根据最近距离匹配)
3. 计算代价函数的最小化解:  $R^* = \underset{R}{\operatorname{argmin}} \sum_n \|p_n - Rq_{\text{match}(p_n)}\|^2$ 
  - ⌚ 最小二乘损失函数,
  - ⌚ 需要额外的约束——R是正交阵(旋转矩阵)
  - ⌚ 使用SVD求解
    - 计算 $3 \times 3$ 矩阵:  $H = \sum_n q_{\text{match}(p_n)} p_n^T$
    - 进行SVD分解:  $H = U \Lambda V^T$
    - 计算得到:  $R^* = VU^T$
4. 修正点云位置:  $q_n \leftarrow R^* q_n$ , 回到步骤2
  - (注意: 一般还需检验 $R^*$ 的行列式是否=1, 确保它是旋转阵)



参考: K. S. ARUN, T. S. HUANG, & S. D. BLOSTEIN, Least-Squares Fitting of Two 3-D Point Sets , IEEE Trans. on Pattern Analysis and Machine Intelligence, 1987

## 程序实现

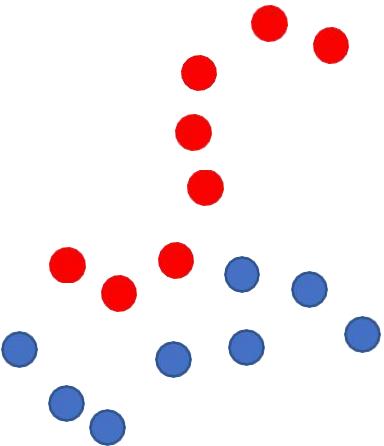
(使用opencv的KNN查询，也可以换成使用PCL实现)

```

7  # 点云匹配
8  # min||(pc1-m1)*R0-(pc0-m0)||
9  def icp(pc0,pc1,it=6,verbose=True):
10     flann = cv2.FlannBasedMatcher(dict(algorithm=1,trees=5),dict(checks=50))
11
12     # 去均值
13     m0=np.mean(pc0,axis=0)
14     m1=np.mean(pc1,axis=0)
15     p=(pc0-m0).astype(np.float32)
16     q=(pc1-m1).astype(np.float32)
17
18     R0=np.eye(3)    # 存放总的旋转矩阵
19     for n in range(it):
20         if True:
21             # 对p的每一点，找到在q中对应点（q中最近距离的点）
22             q_match=np.array([q[m[0]].trainIdx] for m in flann.knnMatch(p,q,k=1)])
23             q_match-=np.mean(q_match,axis=0)
24
25             # 计算旋转矩阵
26             H=np.dot(q_match.T,p)
27             u_,vh=np.linalg.svd(H)
28             R=np.dot(u,vh)
29
30             # 调整点云位置
31             q_=np.dot(q,R)
32             R0=np.dot(R0,R)

```

注意和之前一页  
公式转置关系



```

34     if True:
35         # 对q的每一点，找到在p中对应点（p中最近距离的点）
36         p_match=np.array([p[m[0]].trainIdx] for m in flann.knnMatch(q,p,k=1)])
37         p_match-=np.mean(p_match,axis=0)
38
39         # 计算旋转矩阵
40         H=np.dot(p_match.T,q)
41         u_,vh=np.linalg.svd(H)
42         R=np.linalg.inv(np.dot(u,vh))
43
44         # 调整点云位置
45         q_=np.dot(q,R)
46         R0=np.dot(R0,R)
47
48         if verbose: print('[%d] err: %f' % (n,np.mean(np.abs(p-q))))
49     return R0,m0,m1

```

注意取逆，这是因为需要作用  
在q点云上，而不是p点云

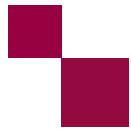


# 目录

## CONTENTS

02

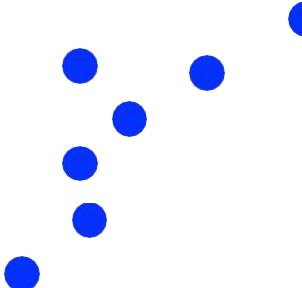
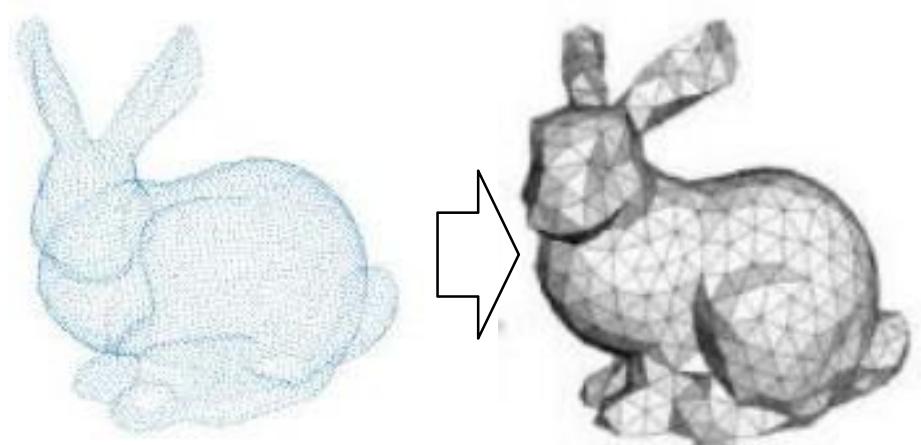
3D表面重建



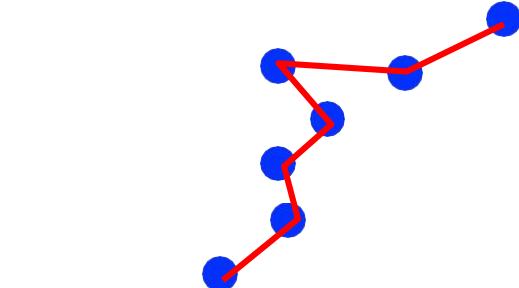
# 3D表面重建



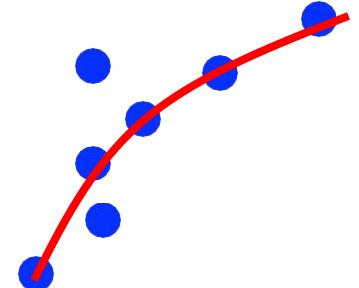
- 三角剖分之前需要获得光滑连续的物体表面
- 如何从点云得到光滑连续的物体表面？
- 难点(看下面例子)
  - 点云之间有空隙和空洞，并不是连续的
  - 点云中有噪声和重叠



:( 如何处理这样的点云，  
物体表面边界在哪里？



:( 糟糕的处理方式，存在  
噪声，虚假的平面褶皱



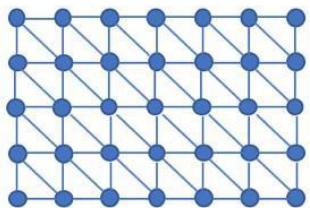
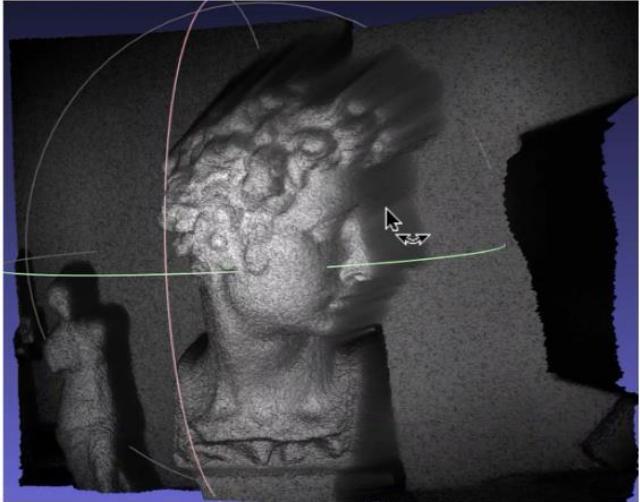
:) 好的处理结果——平滑，连续

# 3D表面重建

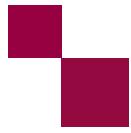


- 简单的方法

- 深度图中按格点直接构造三角形
- 非全方位的完整点云
- 2D距离近邻连接替代了3D距离上的近邻



- 复杂但性能更好的方法——构建SDF函数，然后通过SDF的空间取值计算物体表面位置，并做三角剖分



# 3D表面重建-SDF函数



- SDF函数——signed distance function (SDF)

- 为空间中的每个位置指定一个值 $F(\mathbf{p})$

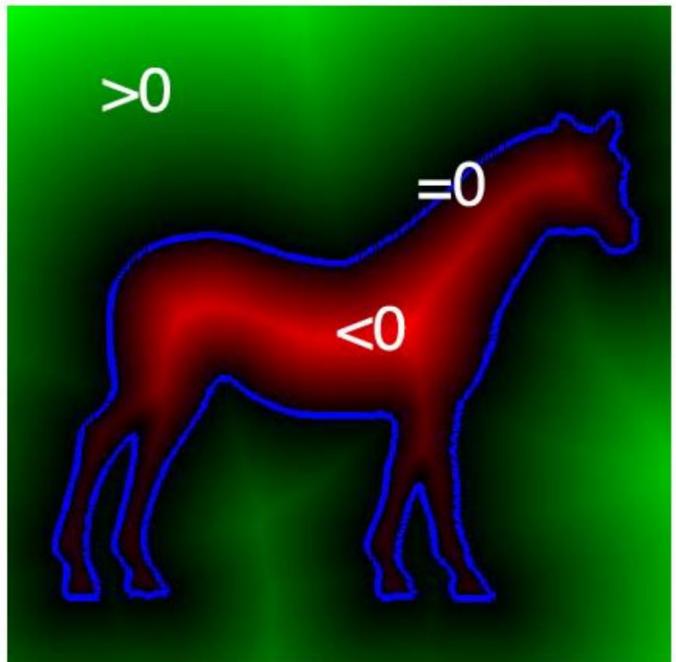
(这里:  $\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ , 代表一个空间位置)

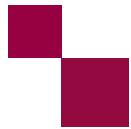
- 曲面把空间分为3部分:

- 曲面内部:  $F(\mathbf{p}) < 0$
- 曲面外部:  $F(\mathbf{p}) > 0$
- 曲面上:  $F(\mathbf{p}) = 0$

- 表面重建

- 构建函数 $F(\mathbf{p})$ 的解析表达式
- 满足 $F(\mathbf{p})=0$ 的空间位置( $\mathbf{p}$ )构成了物体表面





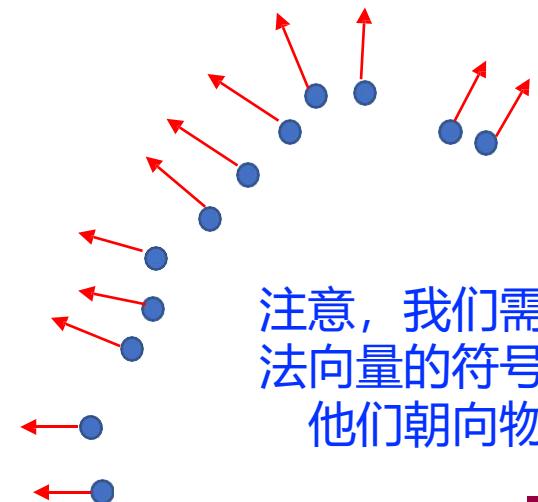
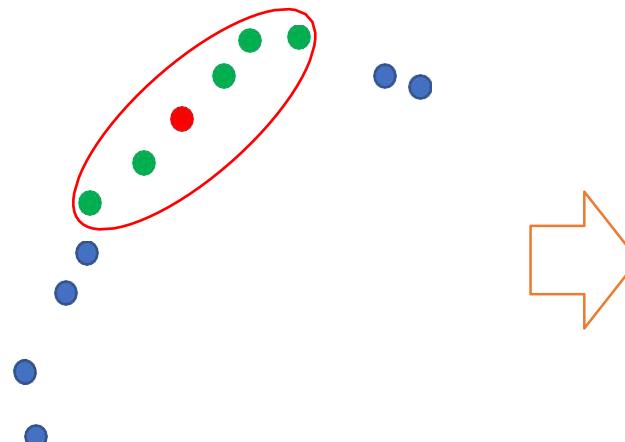
# 3D表面重建-构建SDF函数(RBF)



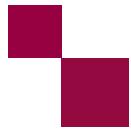
- 如何构建SDF函数?
- 我们利用物体表面的法向量信息构建
- 首先为每个点云上的点计算出他所处的边面的法向量位置

具体计算方法：

1. 对每个点，找到他的K近邻
2. 计算K近邻拟合的平面法向量方向，作为点的法向量方向



注意，我们需要调整法向量的符号，取保他们朝向物体外



# 3D表面重建-构建SDF函数(RBF)



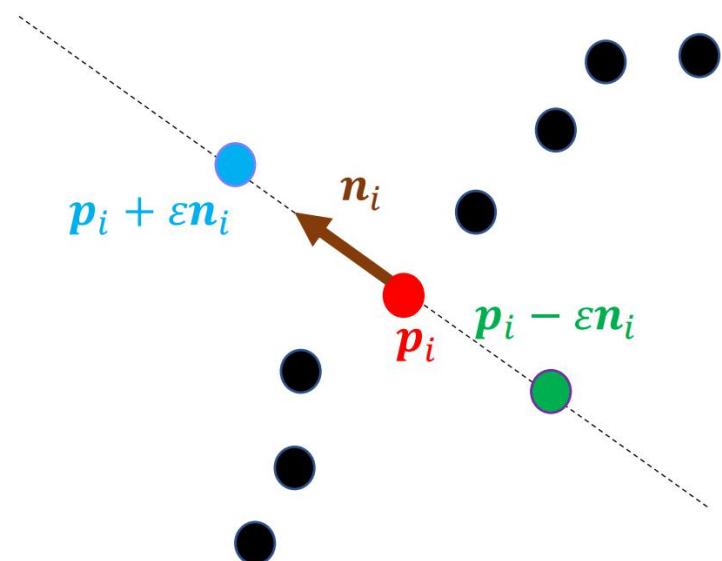
我们构建的SDF函数满足以下条件：

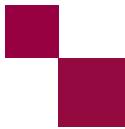
对于每个点 $p_i$ 通过函数 $F(p)$ 的构造，使得：

$$\begin{cases} F(p_i + \varepsilon n_i) = +\varepsilon \\ F(p_i - \varepsilon n_i) = -\varepsilon \end{cases}$$

$\varepsilon$ 是一个相对于点云尺度很小的正数，比如0.1

点 $p_i$ 的法向量 $n_i$





# 3D表面重建-构建SDF函数(RBF)



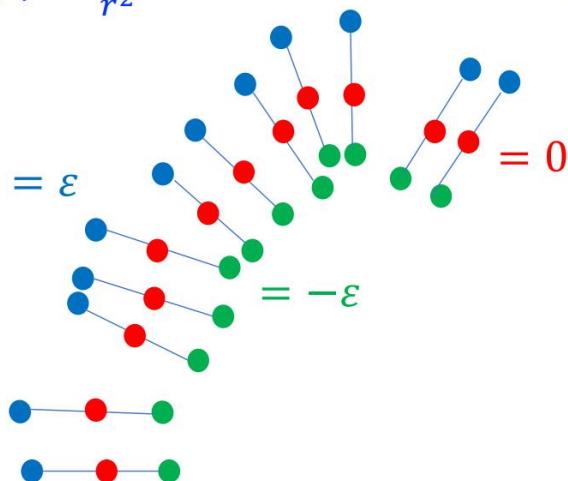
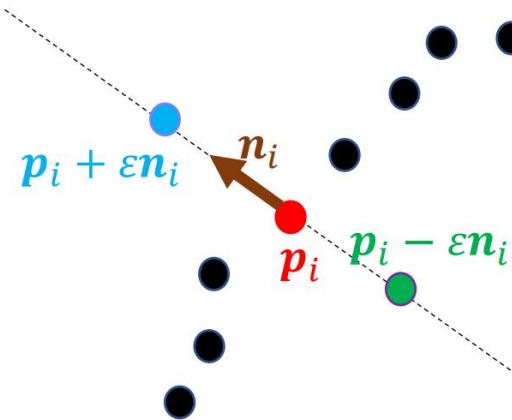
- :( ) •  $F(\mathbf{p})$ 具体表达式是什么?
- : ) • 用一个距离相关的(Radial Basis Function Interpolation)函数的线性组合表示

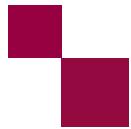
$$F(\mathbf{p}) = \sum_i a_i \phi(\|\mathbf{p}_i - \mathbf{p}\|) + \sum_i b_i \phi(\|\mathbf{p}_i + \varepsilon \mathbf{n}_i - \mathbf{p}\|) + \sum_i c_i \phi(\|\mathbf{p}_i - \varepsilon \mathbf{n}_i - \mathbf{p}\|)$$



$\phi$ 是核函数, 比如:  $\phi(r) = \frac{1}{r^2}$

$\|\mathbf{a} - \mathbf{b}\|$ 表示 $\mathbf{a}$ 和 $\mathbf{b}$ 之间的  
距离 (坐标差的平方  
和的开根号)





# 3D表面重建-构建SDF函数(RBF)



$F(\mathbf{p})$ 具体表达式中，需要待定的是3组系数： $a_i, b_i, c_i$

$$F(\mathbf{p}) = \sum_i \textcolor{red}{a}_i \phi(\|\mathbf{p}_i - \mathbf{p}\|) + \sum_i \textcolor{blue}{b}_i \phi(\|\mathbf{p}_i + \varepsilon \mathbf{n}_i - \mathbf{p}\|) + \sum_i \textcolor{green}{c}_i \phi(\|\mathbf{p}_i - \varepsilon \mathbf{n}_i - \mathbf{p}\|)$$

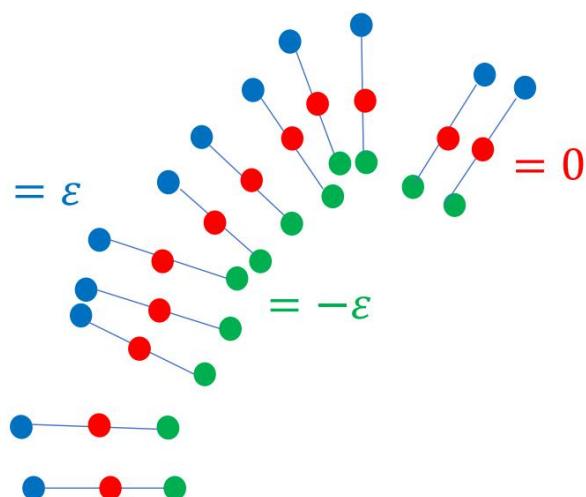


如何求解这3组系数？



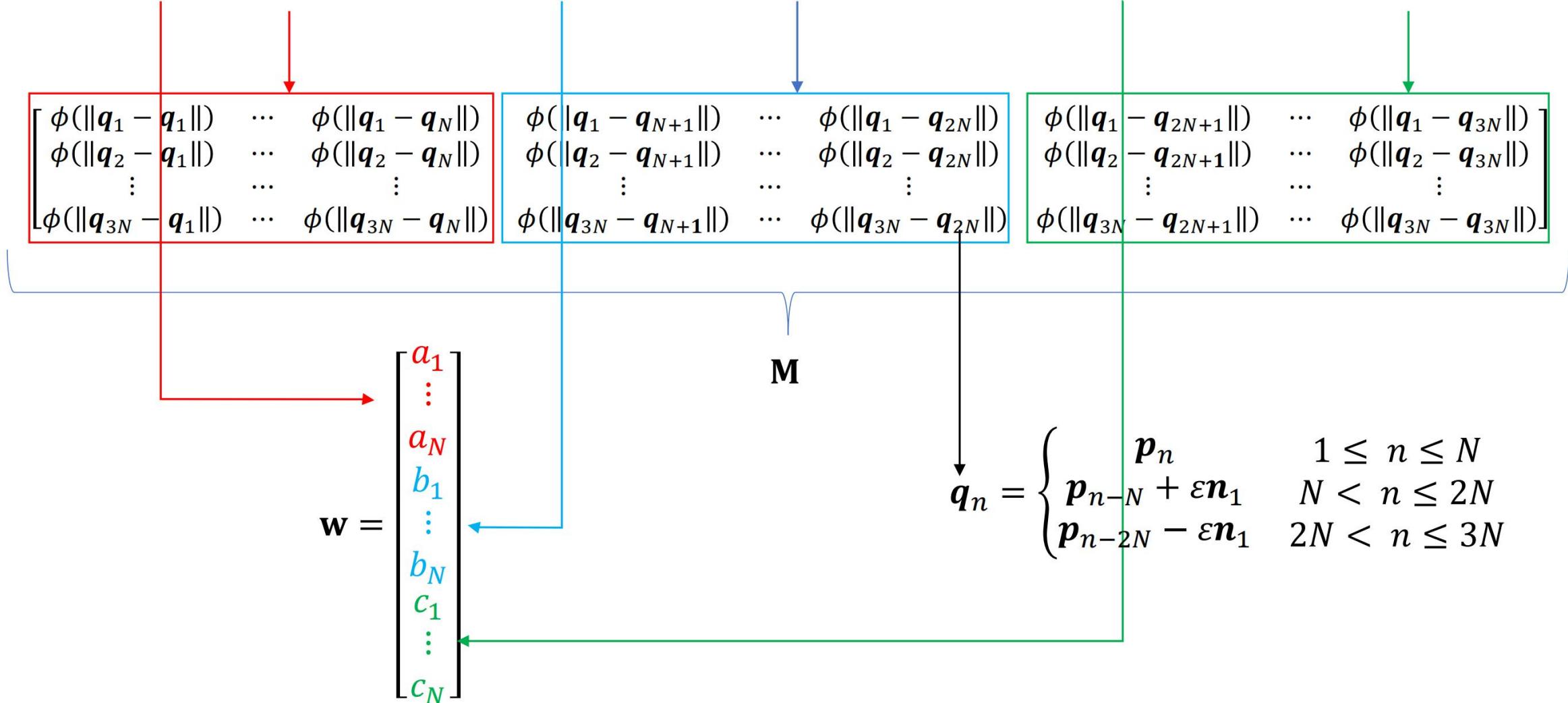
利用：
$$\begin{cases} F(\mathbf{p}_j) = 0 \\ F(\mathbf{p}_j + \varepsilon \mathbf{n}_j) = +\varepsilon \\ F(\mathbf{p}_j - \varepsilon \mathbf{n}_j) = -\varepsilon \end{cases}$$

构成线性方程组，求出他的解



# 3D表面重建-构建SDF函数(RBF)

$$F(\mathbf{p}) = \sum_i a_i \phi(\|\mathbf{p} - \mathbf{p}_i\|) + \sum_i b_i \phi(\|\mathbf{p} - (\mathbf{p}_i + \varepsilon \mathbf{n}_i)\|) + \sum_i c_i \phi(\|\mathbf{p} - (\mathbf{p}_i - \varepsilon \mathbf{n}_i)\|)$$



# 3D表面重建-构建SDF函数(RBF)

$$\begin{bmatrix} \phi(\|q_1 - q_1\|) & \cdots & \phi(\|q_1 - q_N\|) & \phi(\|q_1 - q_{N+1}\|) & \cdots & \phi(\|q_1 - q_{2N}\|) & \phi(\|q_1 - q_{2N+1}\|) & \cdots & \phi(\|q_1 - q_{3N}\|) \\ \phi(\|q_2 - q_1\|) & \cdots & \phi(\|q_2 - q_N\|) & \phi(\|q_2 - q_{N+1}\|) & \cdots & \phi(\|q_2 - q_{2N}\|) & \phi(\|q_2 - q_{2N+1}\|) & \cdots & \phi(\|q_2 - q_{3N}\|) \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \phi(\|q_{3N} - q_1\|) & \cdots & \phi(\|q_{3N} - q_N\|) & \phi(\|q_{3N} - q_{N+1}\|) & \cdots & \phi(\|q_{3N} - q_{2N}\|) & \phi(\|q_{3N} - q_{2N+1}\|) & \cdots & \phi(\|q_{3N} - q_{3N}\|) \end{bmatrix}$$

**Mw = v**

注意到**M**是“方阵”，我们计算这个方程的解：  
 $w = M^{-1}v$

$$w = \begin{bmatrix} a_1 \\ \vdots \\ a_N \\ b_1 \\ \vdots \\ b_N \\ c_1 \\ \vdots \\ c_N \end{bmatrix}$$

$$v = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \varepsilon \\ \vdots \\ \varepsilon \\ -\varepsilon \\ \vdots \\ -\varepsilon \end{bmatrix}$$

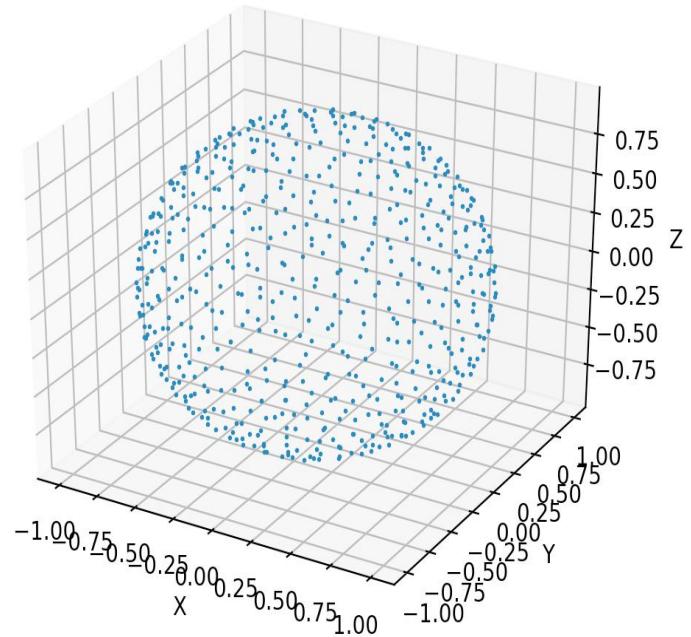
求解矩阵方程得到： $w = M^{-1}v$ ，于是下面构建的SDF函数的加权得到确定

$$F(p) = \sum_i a_i \phi(\|p_i - p\|) + \sum_i b_i \phi(\|p_i + \varepsilon n_i - p\|) + \sum_i c_i \phi(\|p_i - \varepsilon n_i - p\|)$$

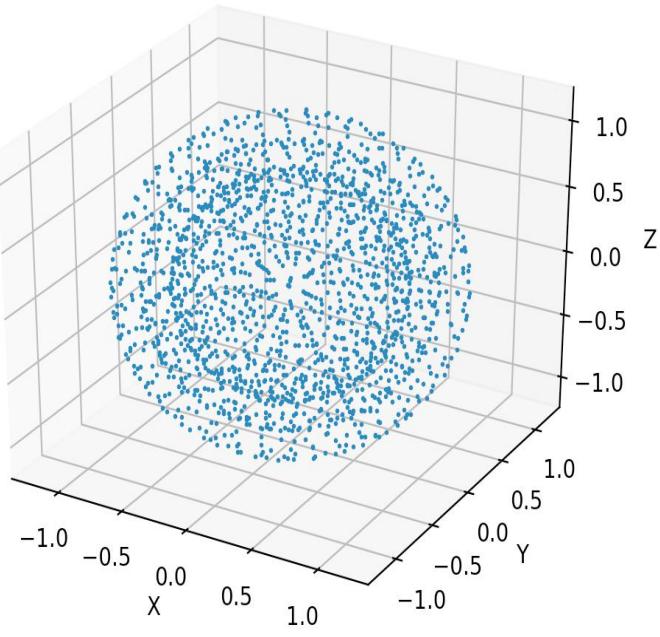
# 3D表面重建-构建SDF函数(RBF)



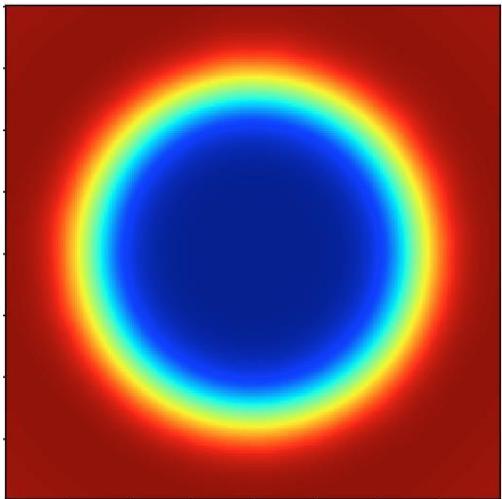
算法流程的实际例子



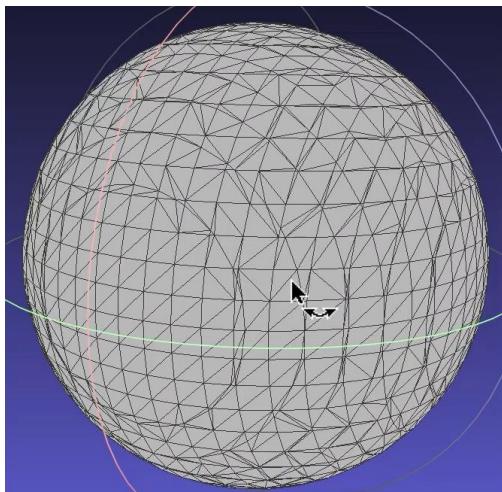
原始离散点云



通过法向量拓展的点云



$z=0$ 的横截面 SDF函数值

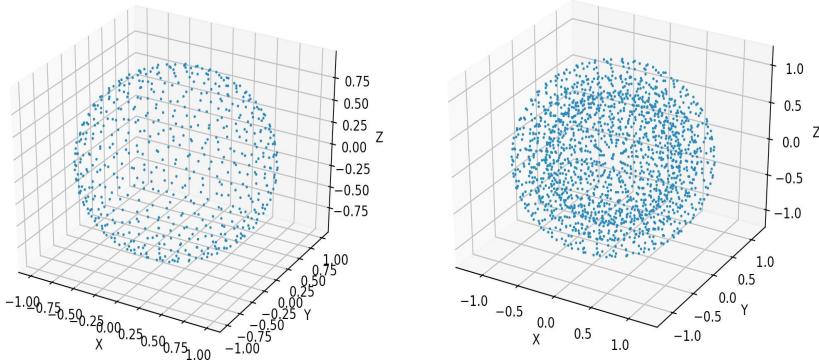


# 3D表面重建-构建SDF函数(RBF)



## 算法流程的代码实现

```
17     ## 点云按法向量扩展
18     def pc_ext(pc,p0,d,k):
19         pcn=find_surf_dir_with_knn(pc,k,func_phi)
20
21         # 法向量方向修正, 这里简化了问题, 仅仅要求法向量背离给定的中心点p0
22         pcn=np.array([n*np.sign(np.dot(n,p-p0)) \
23                         for p,n in zip(pc,pcn)])
24
25         pc_out=pc+pcn*d    # 内点云
26         pc_in =pc-pcn*d    # 外点云
27
28
29         return pc_in,pc_out
```



原始离散点云

通过法向量拓  
展的点云

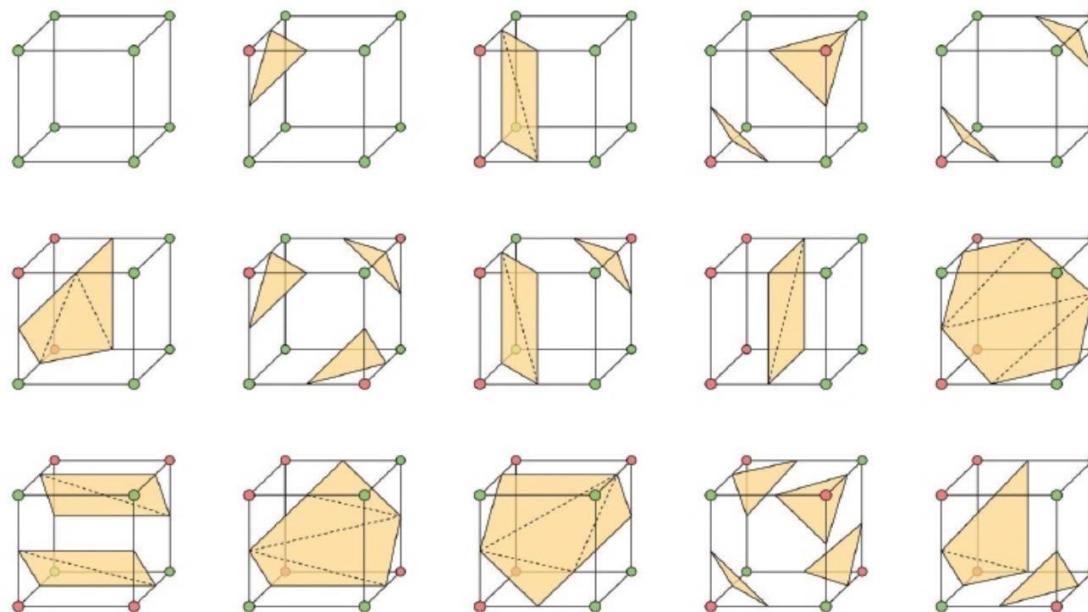
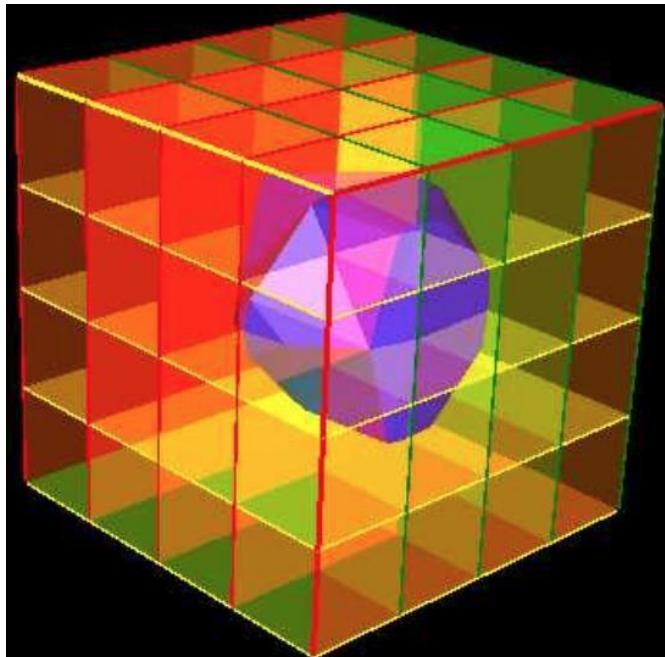
```
1     ## 构建SDF
2     #   pc   点云
3     #   p0   点云内点中心 (用于确定法向量方向)
4     #   d    点云沿法向量正负方向拓展的距离
5     #   k    计算法向量的knn数量
6     def construct_SDF(pc,p0,d,k):
7         # 点云按法向量方向拓展
8         pc_in,pc_out=pc_ext(pc,p0=P0,d=D,k=K)
9         pc_all=np.vstack((pc,pc_out,pc_in))
10
11        # 解线性方程计算SDF
12        M=np.array([func_phi(np_lin.norm(pc_all[r]-pc_all, axis=1)) \
13                     for r in range(3*N)])
14
15        v=np.array([  np.zeros(N),\
16                      D*np.ones (N),\
17                      -D*np.ones (N)]).ravel()
18
19        w=np.dot(np_lin.pinv(M),v).ravel()
20
21        return pc_all,w
```

$$\mathbf{w} = \mathbf{M}^{-1}\mathbf{v}$$

# 3D表面重建

## ——从SDF函数得到物体表面的三角剖分Marching Cubes算法

- 将空间用立方体分割，
- 根据每个立方体的顶点的SDF函数取值正负来确定他是否穿越物体表面，以及如何穿越物体表面的
- 只考虑顶点SDF函数值有正也有负的情况
- 顶点SDF函数值全正或全负时表明立方体整个在物体外部或者内部，不会穿越物体表面，因此不会形成三角形

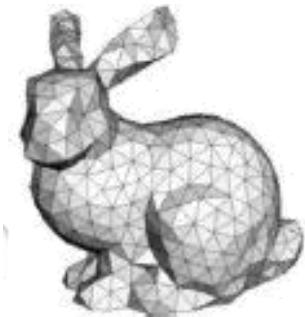


- 考虑8个顶点，共有256种组合，但由于对称/镜像等，可以归并为15种可能，如左图所示
- 图中红色和绿色顶点分别对应不同的符号
- 注意：对特定的顶点符号有“多解”，需要根据额外的检测确定正确的三角形划分

## 柏松重建

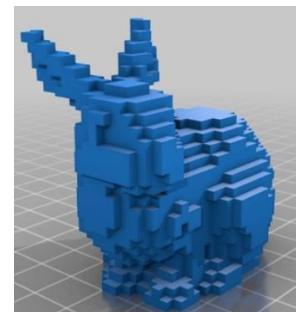
- SDF函数描述表面

- 物体外 $>0$
- 物体内 $<0$
- 物体表面 $=0$

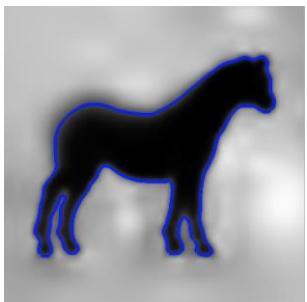


- 指示函数F，描述实体

- 物体外 $=0$
- 物体内 $=1$
- 物体表面 $=0.5$



- 物体表面内外F函数发生突变
- 对F函数改造，使得表面有个“过渡带”
  - 物体外（远离表面） $=0$
  - 物体内（远离表面） $=1$
  - 物体表面附近，由内到外逐步从1过度到0

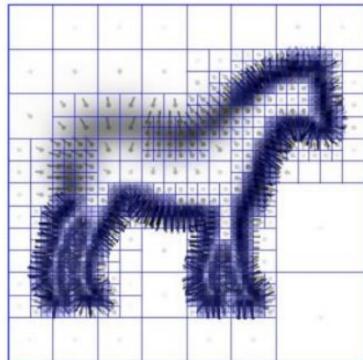


指示函数边界的“糊化”

# 3D表面重建-和物体表面点云局部法向量联系起来

- 指示函数F的梯度（是3维向量）

- 物体内部（远离表面）是0
- 物体外部（远离表面）是0
- 物体表面：非0，指向内部
  - 我们让他等于点云表面的局部法向量（方向指向内部）
  - 法向量只在点云上定义，但我们将其“糊化”一下，扩展到外部附近（距离表面越远，法向量的影响力越小）
  - 把空间“糊化”的法向量记为函数V（“糊化”通过3D滤波可以实现）



法向量（梯度）的“糊化”

- 柏松方程： $\Delta F = \operatorname{div} V$

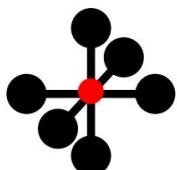
拉普拉斯 $\downarrow$

散度 $\downarrow$ （认为已知）

$$\frac{\partial^2 F_x}{\partial x^2} + \frac{\partial^2 F_y}{\partial y^2} + \frac{\partial^2 F_z}{\partial z^2}$$

$$\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}$$

数值求解得到指示函数F，实际运算时，用有限个参数构建F，用优化求解



$$-(6F_{x,y,z} - F_{x-1,y,z} - F_{x+1,y,z} - F_{x,y-1,z} - F_{x,y+1,z} - F_{x,y,z-1} - F_{x,y,z+1})$$



# 目录

## CONTENTS

03

课堂练习与作业

# 课堂练习



Face1.pcd和Face2.pcd包含了人脸的两部分数据，需要通过ICP算法将脸部数据进行配准

参考：

- ✓ [https://pcl.readthedocs.io/projects/tutorials/en/latest/interactive\\_icp.html#interactive-icp](https://pcl.readthedocs.io/projects/tutorials/en/latest/interactive_icp.html#interactive-icp)

- 1、先对Face1.pcd或者Face2.pcd点云进行平移（通过计算两组点云的中心位置进行平移）
- 2、通过ICP算法迭代进行配准，输出旋转矩阵T
- 3、对合并后的点云进行法向量计算
- 4、通过poission构网算法对脸部模型进行构网，输出PLY文件

