GAMES 102 - 作业 10

彭博

January 16, 2021

1. 本次作业实现 RBF 曲面重建，基于 Python 语言完成曲面重建的主要流程并利用 Open3D 库来进行点云和重建曲面网格的可视化。RBF 曲面重建的原理与 RBF 插值类似，对于空间中的点可以定义场函数 $s(x)$ 如下:

$$s(x) = \sum_{i=1}^{N} \lambda_i \phi_i(x) = \sum_{i=1}^{N} \lambda_i \phi(x - x_i) \tag{1}$$

其中，$\phi_i(x) = \phi(x - x_i) = \exp\{-c_k(x - x_i)^2\}$ 为 RBF 核函数；x_i 为已知空间点坐标，本次作业即为点云坐标；λ_i 为待求解权重。

对于曲面上的点，场函数 $s(x)$ 需要满足：

$$s(x) = \sum_{i=1}^{N} \lambda_i \phi(x - x_i) = 0 \tag{2}$$

因此可以得到 N 个齐次方程：

$$\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_n(x_n) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3}$$

为避免平凡解，需要额外加入约束条件:

$$\sum_{i=1}^{N} \lambda_i = 0 \tag{4}$$

除此之外，RBF 曲面重建还加入了曲面外点来避免非平凡解。设曲面上点 x_i 对应的法向为 n_i，则在法向上场函数需要满足：

$$s(x_i + c \cdot n_i) = c \tag{5}$$

$$s(x_i - c \cdot n_i) = -c \tag{6}$$

因此，场函数 $s(x)$ 实际上表示的是曲面的有向距离场；RBF 曲面重建的实质就是用 RBF 函数来逼近该有向距离场。对以上 $3N + 1$ 个方程进行求解即可得到所需的场函数，整个重建算法流程如下：

(a) 读取点云文件并计算点云法向;
(b) 将点云坐标带入场函数，得到 N 个齐次方程;
(c) 将曲面外点坐标带入场函数，得到 2N 个方程;
(d) 将系数的约束加入方程组;
(e) 求解 3N+1 个方程，得到 RBF 核函数对应系数;
(f) 使用 Marching Cubes 算法提取场函数的 0 等值面，即为重建后的曲面。

使用 RBF 曲面重建的核心代码可参见 Listing 1，重建后的点云曲面可参见 Fig.1-Fig.3:

Listing 1 RBF 曲面重建

```python
def RBF_Reconstruction(pcl, c=1e-5, kc=1.0):
    """
    3D Reconstruction with RBF functions.

    Args:
        pcl: a point cloud;
        c: the distance parameter;
        kc: the kernel parameter;

    Returns:
        w: the weights vector;
    """

    ## retrieve points and normals from point cloud
    points = np.asarray(pcl.points)
    normals= np.asarray(pcl.normals)

    ## initialization
    N = points.shape[0]

    A = np.zeros((3*N+1, N))
    b = np.zeros(3*N+1)

    ## surface points
    print("Adding surface points ...")

    A[:N, :N] = np.exp(-kc * vecNorm(points, points))

    ## off-surface points
    print("Adding off-surface points ...")

    A[N:2*N, :N] = np.exp(-kc * vecNorm(points + c*normals, points))
    b[N:2*N] = c

    A[2*N:3*N, :N] = np.exp(-kc * vecNorm(points - c*normals, points))
    b[2*N:3*N] = -c

    ## constraints on weights
    A[-1,:] = 1

    ## solve the equation
    print("Solving the equation ...")
    w = linalg.lstsq(A, b)[0]

    return w
```

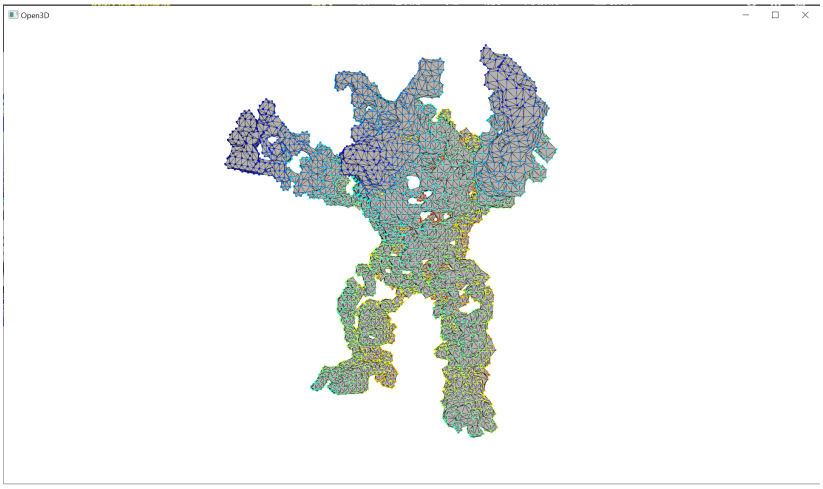

Figure 1: Arma 曲面重建

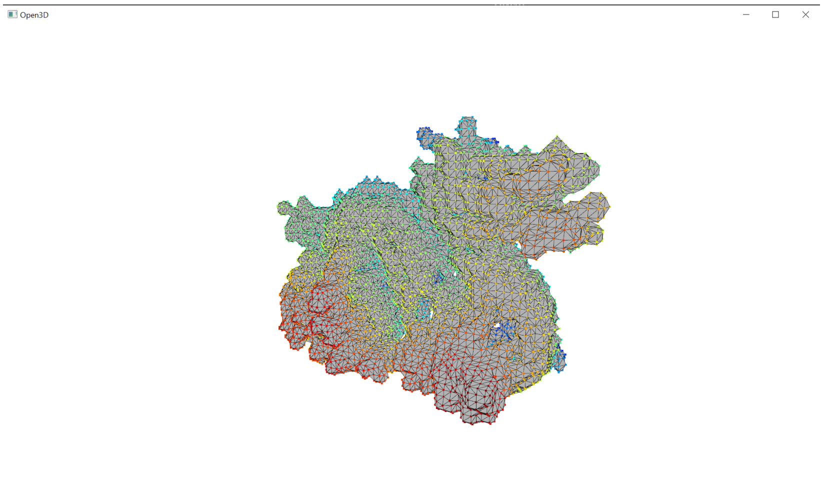

Figure 2: dragon 曲面重建

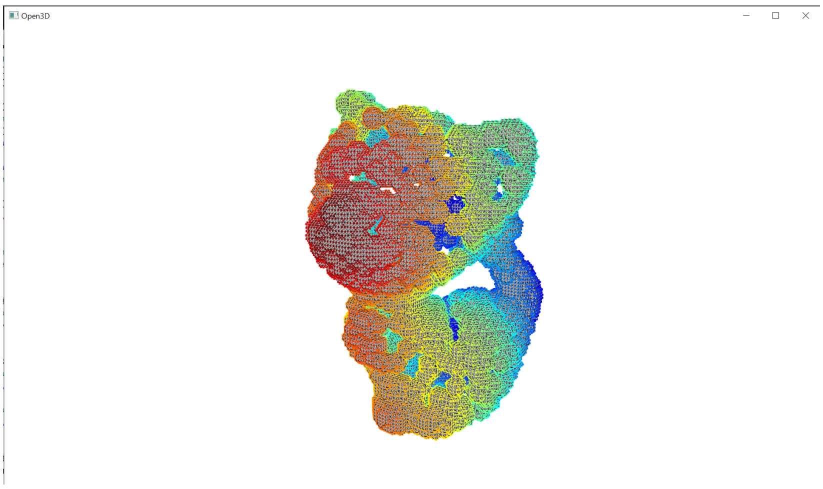

Figure 3: kitten 曲面重建

本次作业中一些值得说明的问题：

(a) 本次作业涉及到大规模的矩阵运算，因此在 python 中需要使用到向量化的技巧，直接使用循环操作会使程序的运行效率非常低;

(b) 求解 RBF 核函数权重时需要求解稠密线性方程组，求解效率要比求解稀疏方程低很多;

(c) RBF 重建需要知道点云的法向，本次作业中使用 Open3D 库来估计点云法向。将法向进行可视化后可以发现有些位置的法向是错误的，因此若是提前知道点云的法向则可以获得更好的重建效果;

(d) RBF 重建对核函数参数以及距离参数比较敏感，本次作业 3 个模型均采用了不同的参数设置才能获得较为理想的结果；

(e) 由于使用了向量化的技巧使得本次作业对内存的需求较高，3 个模型均是在 4% 采样的点云模型进行重建，若是使用其他加速技巧则可以考虑使用更为稠密的点云；

(f) 除了场函数的参数外 Marching Cubes 算法的相关参数也会对曲面重建结果产生影响，本次作业重建曲面上的孔洞可能是来源于 Marching Cubes 算法对空间采样不足；