几何计算前沿第二次作业报告
Laplacian Mesh Smoothing 的实现

沈千帆 2200013220

2024 年 4 月 20 日

1 简述

本作业主要在`main.py`中实现了利用余切算子的拉普拉斯网格平滑算法，整个算法可以分为以下几个部分：

- `def load_obj(filename)`: 这个函数实现了原始 mesh 文件的加载。
- `def cal_angle_cot(v1, v2, v_angle)`: 计算夹角的 cot 值。
- `def cal_area(tm, idx)`: 计算每个顶点作为顶点所有的三角形的面积之和。
- `def cal_laplacian_weight(tm)`: 计算使用余切算子的拉普拉斯矩阵
- `def smooth(tm, iterations, method= 'explicit', lam = 1e-2)`: 利用拉普拉斯矩阵进行迭代平滑，可以选用 **explicit** 或者 **implicit** 迭代。

2 具体实现

2.1 拉普拉斯矩阵计算

根据公式

$$\Delta f(v_i) := \sum_{v_j \in N(v_i)} w_{ij}(f_j - f_i)$$

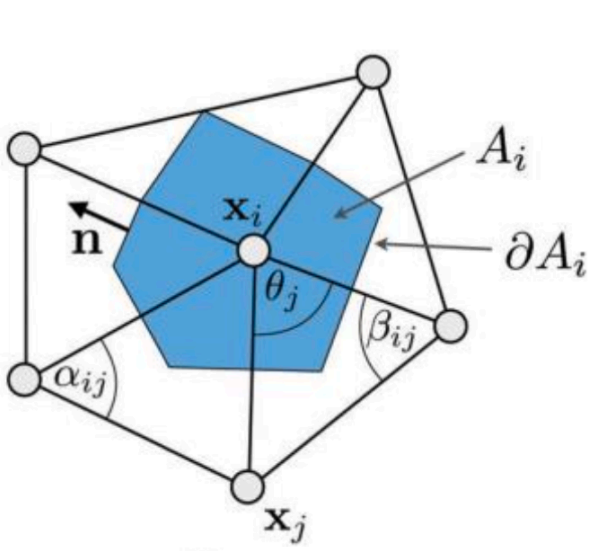
如下图，对于每个顶点 v_i，其偏移量为相对于其周围顶点的便宜乘上权重求和。而我们在这个算法中使用的权重是余切算子，因此权重可以用余切算子构成拉普拉斯矩阵 L，其中

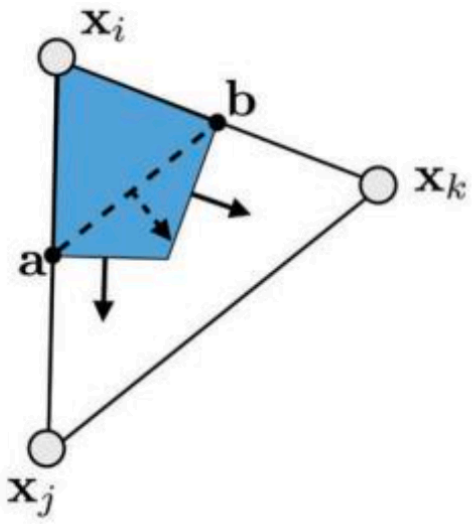
$$L_{ii} = -\sum_j w_{ij}, L_{ij} = w_{ij} = \frac{1}{2A_i}(\cot \alpha_{i,j} + \cot \beta_{i,j})$$

因此在实现的时候我们将 L 用 M 和 C 来表示：

```
M = np.zeros((n, 1)) # calculate each vertice's triangle area sum
C = np.zeros((n, n)) # calculate each vertice's cot weight
L = C / M
L = scipy.sparse.csc_matrix(L)
```

在这里为了加速使用了**稀疏矩阵**的函数。





2.2 网格平滑

根据拉普拉斯矩阵的性质，此时每个顶点 v_i 的偏移可以表示为 $\Delta V = L \cdot V$，每次迭代有两种方式，在本作业中**分别进行了实现**。对于**显式**即 $V^k = V^{k-1} + \lambda L \cdot V^{k-1}$，**隐式**则需要解方程 $(I - \lambda L)V^k = V^{k-1}$。更详细地在`def smooth(tm, iterations, method, lam):`中：

```
for i in range(iterations):
    vertices += lam * L @ vertices # 显式迭代
    A = scipy.sparse.csc_matrix(I - lam * L) # 隐式迭代
    vertices = spsolve(A, vertices) # 同样应用稀疏矩阵求解
```

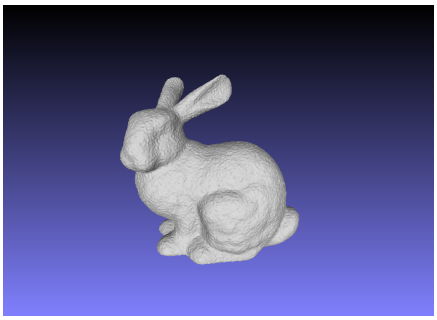
特别地，这里的学习率选择 $1e-2$ 实现效果较优，同时在实验的过程中发现面积需要乘上相应的比例，否则会导致数量级爆炸。

2.3 可视化

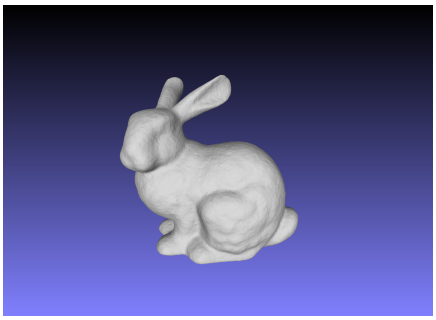
在计算拉普拉斯矩阵和迭代的时候，分别调用`tqdm`库进行可视化，可视化效果如下：

```
Calculation Progress: 100%|██████████| 29849/29849 [02:22<00:00, 209.17it/s]
Smoothing Progress(iterations = 5): 100%|██████████| 5/5 [00:01<00:00,  3.47it/s]
Smoothing Progress(iterations = 10): 100%|██████████| 10/10 [00:03<00:00,  3.04it/s]
Smoothing Progress(iterations = 30): 100%|██████████| 30/30 [00:09<00:00,  3.25it/s]
Smoothing Progress(iterations = 50): 100%|██████████| 50/50 [00:17<00:00,  2.93it/s]
Smoothing Progress(iterations = 80): 100%|██████████| 80/80 [00:22<00:00,  3.48it/s]
```

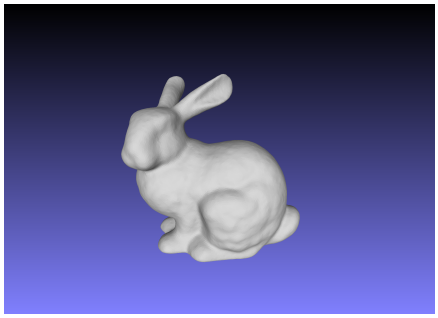
3 实现结果



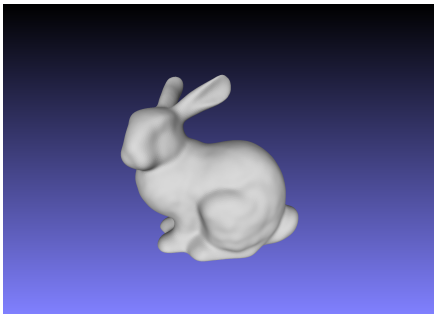
(a) 原始



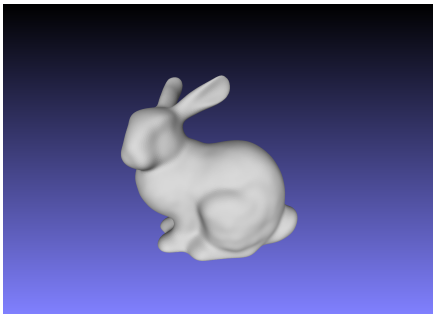
(b) iterations=5



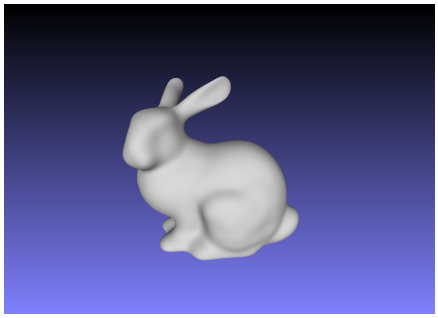
(c) iterations=10



(a) iterations=30



(b) iterations=50



(c) iterations=80