Lab2-MeshSmoothing

王想 2100013146

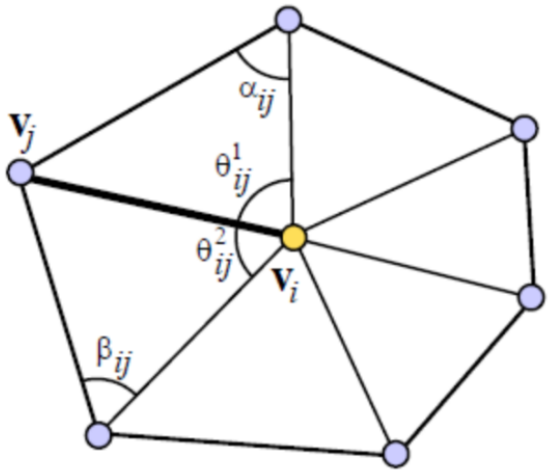
1 算法实现

算法大致流程如下：

1. 对每个顶点 v_i ，计算邻居位置的加权平均 $v_i^* = \frac{\sum_{j\in N(i)} w_{ij}v_j}{\sum_{j\in N(i)} w_{ij}}$ 其中，$w_{ij} = \cot\alpha_{ij} + \cot\beta_{ij}$, 夹角 α_{ij}, β_{ij} 见下方示意图。

2. 更新顶点：$v_i = (1-\lambda)v_i + \lambda v_i^*$

3. 重复 1-2 步，直到迭代次数达到上限

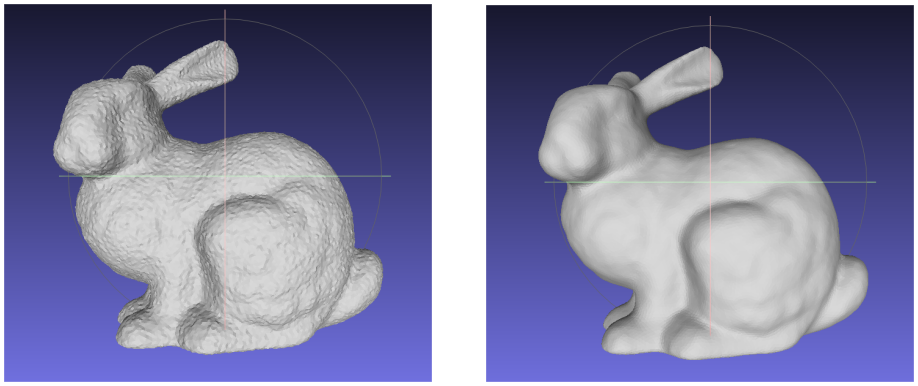


需要注意的是这里需要在每一次迭代之前先把更新前顶点的位置复制一份保存下来，在后续更新的计算中，邻居点的位置应该是更新前的邻居顶点的位置。如果没有留存更新前的顶点坐标就直接改变 mesh 的顶点坐标，会导致用新的顶点坐标去更新其他点的坐标，这些微小的偏差会积累起来导致最后的平滑效果出现较大误差。

2 Results

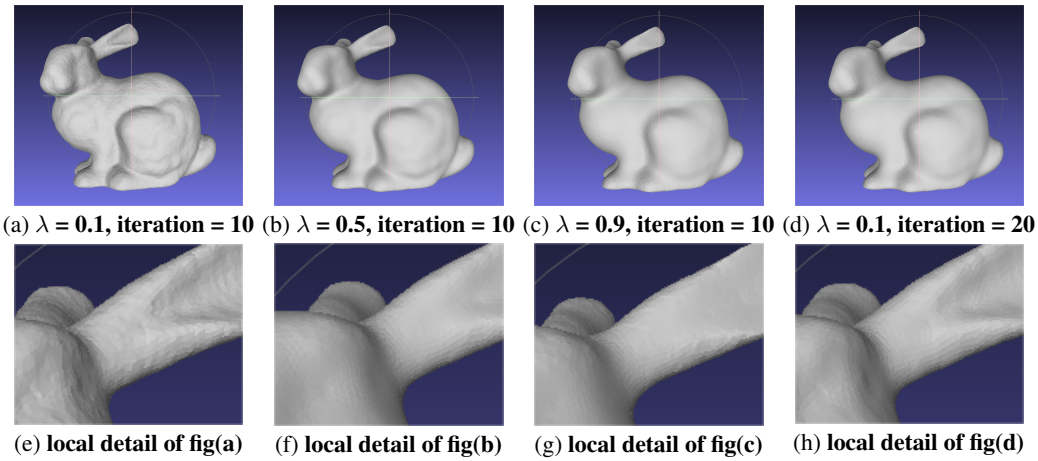
迭代步长 λ 越大，平滑程度越大，需要的迭代次数也越小，但相应的缺点是容易过度平滑；迭代次数 $iteration$ 越大，平滑程度越大，但代价是时间消耗过多，在我的电脑中串行执行时，迭代一次平均花费 21s。在尝试多组超参之后，我发现小步长、大迭代次数的效果最好。

图 2 为原网格与平滑后网格的对比，图 3 为超参 λ 的调整对平滑效果的影响对比 (这里只展示部分有代表性的超参)。可以看到，在参数 $\lambda = 0.1, iteration = 10$ 下，平滑程度不太够，仍显得有点杂乱；在参数 $\lambda = 0.5, iteration = 10$ 下，兔耳已经被磨平了一部分；在参



(a) **Origin** (b) λ **= 0.1, iteration = 20**

图 1: **Smooth**



(a) λ **= 0.1, iteration = 10** (b) λ **= 0.5, iteration = 10** (c) λ **= 0.9, iteration = 10** (d) λ **= 0.1, iteration = 20**

(e) **local detail of fig(a)** (f) **local detail of fig(b)** (g) **local detail of fig(c)** (h) **local detail of fig(d)**

图 2: **Hyperparamerter Tuning**

数 $\lambda = 0.5, iteration = 10$ 下，兔耳被完全磨平；在参数 $\lambda = 0.1, iteration = 20$ 下，既保证了平滑效果，又防止了过度平滑。

3 Improvements

在保存下每一轮迭代更新前顶点的坐标后，考虑到每一个顶点的更新计算均是平行、独立的，不会相互影响，于是我们可以采用并行计算分别对每一个点进行更新，这样会大大加速平滑计算，相较于串行计算能节省大量时间。

A Appendix

Requirement: $openmesh$, $numpy$

Running command: $python\ smooth.py$