计算机图形学大作业 1 项目报告
网格分割

夏卓凡　2020211000　自硕 20

2020 年 12 月 15 日

1　程序框架说明

本文根据文章《*Hierarchical Mesh Decomposition using Fuzzy Clustering and Cut*》[1]，实现了网格分割的 k 路分解算法。程序将这一算法实现为如下几个步骤：定义对偶网络及其权重，求解距离矩阵，初始化 k 个中心点并迭代聚类，对模糊区域构建运输网络并用最小割细化分割结果。接下来我们将分别介绍这些步骤的具体执行流程。

1.1　对偶网络的定义

假设面片模型中仅包含一个连通区域 F，我们定义 F 上相邻面片 f_i, f_j 的测地距离和角度距离

$$\mathrm{Geod}(f_i, f_j) = |P_1S| + |P_2S|, \tag{1}$$

$$\mathrm{Angd}(f_i, f_j) = \eta(1-\cos(<n_1, n_2>)), \tag{2}$$

其中 P_1, P_2 为 f_i 和 f_j 的重心，n_1, n_2 为 f_i 和 f_j 的法向量，S 为将 f_i 和 f_j 展开到平面情况下 P_1P_2 连线与 f_i 和 f_j 相交直线的交点，如图 1 所示。求角度距离，如果面心连线与某个面法向夹角为钝角则 $\eta = 0.2$（凸），否则 $\eta = 1.0$（凹）。求测地距离，可利用图 2 所示几何关系，分别连接 P_1 和 P_2 和 f_i 与 f_j 的公共边上顶点 T，那么当 P_1, P_2, T 共面时 $|P_1P_2|$ 即为测地距离，根据向量内积关系

$$\alpha = \arccos\left(\frac{\overrightarrow{TP_1}\cdot\overrightarrow{TS}}{|P_1T||ST|}\right), \tag{3}$$

$$\beta = \arccos\left(\frac{\overrightarrow{TP_2}\cdot\overrightarrow{TS}}{|P_2T||ST|}\right), \tag{4}$$

求出 α 和 β 后再根据余弦定理，可以计算测地距离的平方

$$(|P_1S|+|P_2S|)^2 = |P_1T|^2 + |P_2T|^2 - 2|P_1T||P_2T|\cos(\alpha+\beta). \tag{5}$$

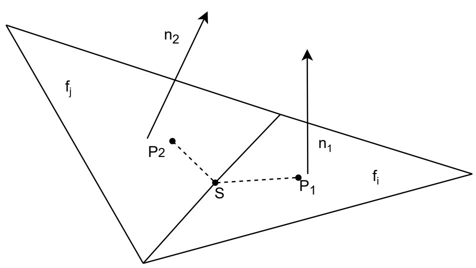

图 1: 相邻面片及其法向量

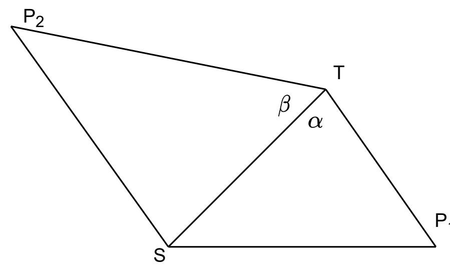

图 2: 测地距离计算

接下来，对对偶网络中的不同节点 i,j，如果它们对应的面片相邻，则按照文章中定义权重，否则设为 ∞，公式为

$$w_{ij} = w_{ji} = \delta \cdot \frac{\mathrm{Geod}(f_i, f_j)}{\frac{1}{|E|}\sum_{e_{ij}} \mathrm{Geod}(f_i, f_j)} + (1-\delta) \cdot \frac{\mathrm{Angd}(f_i, f_j)}{\frac{1}{|E|}\sum_{e_{ij}} \mathrm{Angd}(f_i, f_j)}, \tag{6}$$

其中分母部分的 E 表示对偶图中的邻接节点，分母表示对所有对偶图中的边求平均。

1.2 求解距离矩阵

在获得了初始权重矩阵 W 之后，接下来我们计算对偶网络中任意两节点的最短距离，也就是聚类中用到的距离矩阵。这是一个全源最短路问题，应用 Floyd-Warshall 算法可以求解，如算法 1 所示。然而，算法关于面片数 n 的时间复杂度为 $O(n^3)$，这会导致运行时间变得相当长。为了快速地计算出大规模网络的最短路径，我们使用 GPU 加速，借用了一份 CUDA 上的 Floyd-Warshall 算法的实现代码[1]。

Algorithm 1 Floyd-Warshall 算法

Require: 网络权重矩阵 W，节点数 n

Ensure: 任意两节点间最短路径矩阵 V

```
V ← W
for k = 1...n do
    for i = 1...n do
        for j = 1...n do
            V_ij ← min(V_ij, V_ik + V_kj)
        end for
    end for
end for
```

[1] https://github.com/MTB90/cuda-floyd_warshall

1.3 初始化中心点与聚类

在得到了对偶网络的各个面片间的最短路径矩阵 V 后，我们按照文章中对 k 路分解的说明初始化。文章对第一个初始中心点，选择距离其他点距离之和最大的点，而之后的中心点则最大化该点到当前中心点集的距离。考虑到与 $0-1$ 分解的一致性，而且 $k \geq 2$，我们在实现上先取距离最远的一对点作为前两个中心点，之后再随着 k 的增大，最大化各个中心点到当前中心点的集合的距离。事实上，我们如果定义集合外点到集合的距离为点到集合内各点的最小距离，则选取的初始中心点与文章中一致。在选择好中心后，我们继续初始化每个节点属于各个中心点的概率矩阵 P，其中节点 f_i 属于中心 c_j 的概率为 $P_{ij} = \frac{1/V_{ij}}{\sum_{j=1}^{k} 1/V_{ij}}$，其中分母表示对 k 个中心点的距离倒数求和，保证属于各类的概率值求和为 1。在这里我们为了采取文章中改进的概率更新方式，定义一个辅助集合 A 用于表示各个面片被分配到的中心点

$$A_c = \{i | \max_{j=1,\ldots,k} P_{ij} = c, i = 1,\ldots,n\},\ c = 1,\ldots,k. \tag{7}$$

则 A 为所有 A_c 组成的集合。在迭代聚类的步骤中，更新各个节点的从属概率和更新聚类中心交替进行，当达到最大迭代步数或聚类中心不变后停止运行，如算法 2 所示。

Algorithm 2 迭代模糊聚类算法

Require: 节点数 n，距离矩阵 V，类别数 k，中心点集合 C，分配 A，概率矩阵 P，最大迭代步数 M

Ensure: 更新后的中心点集合 C, 更新后的概率矩阵 P

```
for i = 1,...,M do
    for j = 1,...,n do
        for c = 1,...,k do
            d_jc ← (1/|A_c|) Σ_{l∈A_c} V_jl
            P_jc ← (1/d_jc) / (Σ_{c=1}^{k} 1/d_jc)
        end for
    end for
    for c = 1,...,k do
        C'_c ← min_{i=1...n} Σ_{j=1...n} P_jc V_ji
    end for
    按公式 (7) 更新 A
    if C' = C then
        break
    end if
    C ← C'
end for
```

1.4 模糊区域的选择与分割结果的细化

直接取最大概率的中心点作为各面片的归属往往导致不整齐的边缘，这是由于在模糊区域相当多的中心具有不相上下的概率。设对面片 f_i 而言，最大概率和第二大概率为 P_{i,c_1}, P_{i,c_2}，如果给定模糊阈值 ε，$P_{i,c_1} - P_{i,c_2} \le \varepsilon$，则认为面片 f_i 需要进一步细分，否则可以确定其标签为 c_1。

对 k 类的情况，遍历所有面片，如果需要细分，则按照其类别放入候选集合中；之后我们遍历所有需要细分的两类，按照文中所述的方式添加需要细分的面片外一圈相邻的两类面片，并添加虚拟的原点、汇点。对每个类别对，分别应用 Ford-Fulkerson 算法，求出最小割，如算法 3 所示。事实上，在最大流算法的最后一次迭代后，所有前驱为空的节点和非空的节点自然就是所求的割。

Algorithm 3 Ford-Fulkerson 最小割生成算法

Require: 增广流量矩阵 G，流量图节点数 m，源点 S，汇点 T

Ensure: 最大流前驱向量 p

```
function BFS(s, t)
    p ← {−1, ..., −1}, f ← +∞, q ← queue(s)
    while !q.empty() do
        u ← q.pop()
        if u = t then
            break
        end if
        for i = 1, ..., m do
            if i ≠ s, G_ui > 0, p[i] = −1 then
                p[i] ← u, f ← min(G_ui, f), q.push(i)
            end if
        end for
    end while
    return −1 if p[t] = −1 else f
end function
while Δf = BFS(S, T) > 0 do
    k ← T
    while k ≠ S do
        l ← p[k], G_lk ← G_lk − Δf, G_kl ← G_kl + Δf, k ← l
    end while
end while
```

2 测试结果

图 3: Arma 模型（2400 面片）$k = 8$ 时的分割结果

图 4: dinosaur.2k 模型（4000 面片）$k = 10$ 时的分割结果

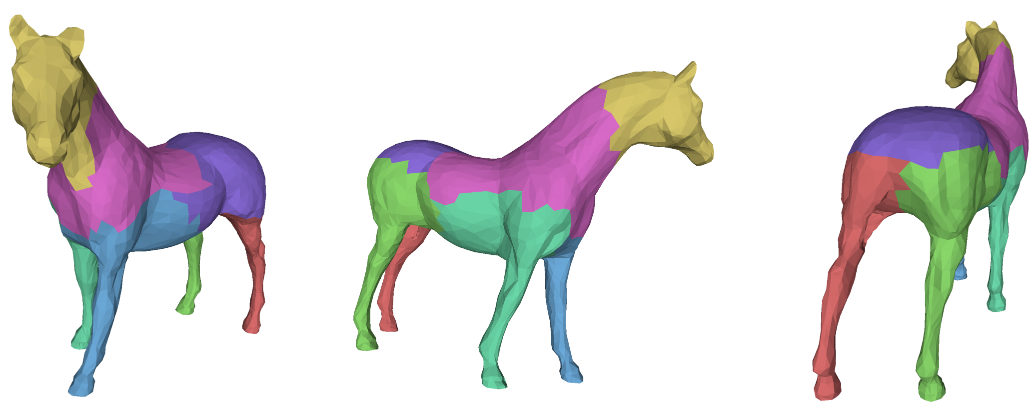

图 5: horse.fine.90k 模型（4000 面片）$k = 7$ 时的分割结果

3 误差与性能分析

3.1 实现细节

源代码主要分为 3 个部分，分别对应本文中的 3 个算法。类 `GeoGraph` 对应了对偶网络的创建和最短距离，其中调用了 GPU 加速的 Floyd-Warshall 算法（`APSPCuda.cuh`、`APSPCuda.cu`）；类 `FuzzyCluster` 实现了类中心点的初始化和迭代聚类算法；类 `Mincut` 实现了基于最小割的细分调整。其他的文件中，`common.h` 和 `common.cpp` 实现了一些通用的各类距离计算函数以及调色板的生成；`main.cpp` 为主程序，为了运行方便，我们使用了一个命令行参数处理工具 `argparse.hpp`[2]。

程序的开发环境为 Visual Studio 2019 (C++ latest)，OpenMesh 8.1 以及 CUDA 10.1，运行环境为 Windows 10 2004，CPU Intel i7-8700K，内存 32GB，GPU NVIDIA GeForce GTX 1080 Ti。

3.2 超参数分析

η 的影响 η 作为角度距离计算中重要的参数，影响着面片对偶网络的距离和最小割的容量。我们选取模型 horse.fine.90k 进行 $\delta = 0.8, k = 6, \varepsilon = 0.05$ 的分割实验，分别观察 $\eta = 0.01, 0.2, 1.0$ 的分割结果，如图 6 所示。η 表示对凸区域的抑制作用，η 越小，相邻面片呈凸角度时角度距离就越小，而呈凹角度时不变。观察分割结果，当 $\eta = 0.01$ 非常小的时候，分割的角度距离受控于凹区域，即在模型表面曲率变化较大的部分产生边界，且锯齿较多；而当 $\eta = 1.0$ 较大的时候，分割区域的边界相对平滑，但语义信息不再明确，如马的两条后腿分割的区域对称性较差；而 $\eta = 0.2$ 是一个权衡的方案，能在保证分割尽量利用凹区域的同时减少锯齿。

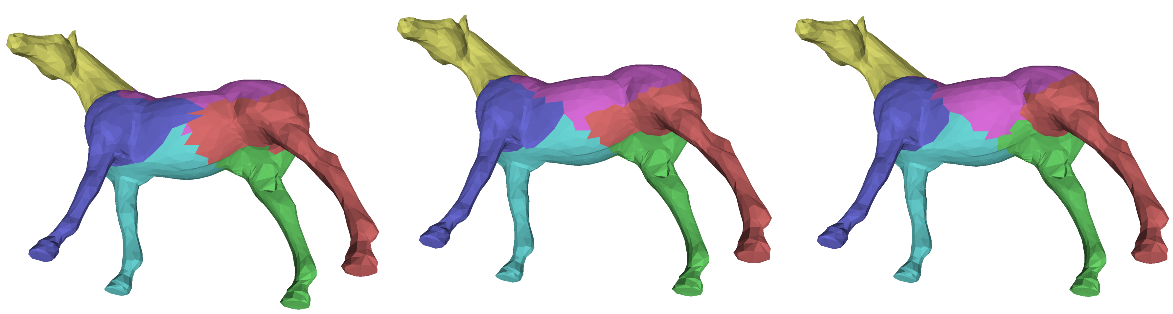

图 6: horse.fine.90k 模型 $\eta = 0.01, 0.2, 1.0$ 时的分割结果

δ 的影响 δ 为平衡测地距离和角度距离的因子，$\delta = 0$ 时距离矩阵只考虑角度距离，$\delta = 1$ 时距离矩阵只考虑测地距离。如图 7 所示，对 $k = 5$ 的 dinosaur.2k 模型，当 $\delta = 0$ 时，分界面对左右对

[2]https://github.com/p-ranav/argparse

称的部分不敏感，容易分成一部分，而增大 δ 后，测地距离的引入让分割结果的语义特征更明确，将恐龙的左右两条腿分别分到了不同部分。

图 7: dinosaur.2k 模型 $\delta = 0.0, 0.5, 1.0$ 时的分割结果

ε 的影响 我们主要观察 ε 对模糊区域大小的影响，图 8 展示了不同 ε 值下 block 模型的模糊区域，其中亮色区域表示原文中的 V_{CA} 和 V_{CB}，浅灰色区域表示 V_C，当 ε 增大时，需要执行最小割细分的区域变大，但在一定范围内 ε 的变化仍能得到相对稳定的最终分割结果。

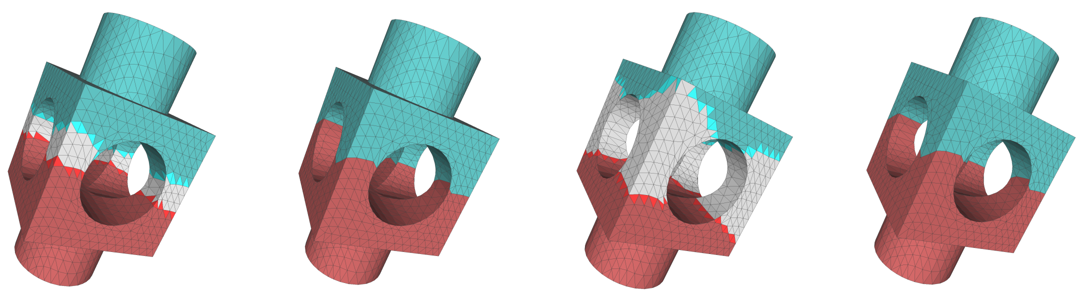

图 8: block 模型 $\varepsilon = 0.05, 0.15$ 时的分割结果

3.3 运行时间分析

这一部分主要展示 CUDA 加速的 Floyd 算法的效率提升，所有时间均在 Release 模式下（O2 优化）测试，Debug 模式下 CPU 版 Floyd-Warshall 算法无法在合理时间内结束。运行时间结果如表 1 所示，单位为毫秒（ms）。

从表中可以看出，程序运行速度的瓶颈在 $O(n^3)$ 的最短路径算法上，因此使用 GPU 优化十分有必要，而且优化后其运行速度提升非常显著。另外，由于迭代聚类算法的复杂度为 $O(kn^2)$，在类别数较多的情况下，其运行时间也会相对变长。而第三部分的复杂度为 $O(m^2)$ 且模糊部分的节点数仅占全部节点的很小一部分，$m \ll n$，因此这一部分的运行速度最快。

模型	k	面片数	算法1运行时间	算法2运行时间	算法3运行时间	整体运行时间
Arma	6	2400	259 / 12251	216 / 219	7 / 7	497 / 14208
horse.fine.90k	7	4000	600 / 55003	647 / 650	22 / 23	1272 / 55680
block	2	4272	646 / 67145	196 / 200	34 / 34	880 / 67383
bunny.fine	6	8000	2822 / 436274	4067 / 3946	140 / 140	7033 / 440365

表 1: 算法各部分运行时间

4 总结

在本次大作业中，我尝试根据《*Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts*》复现了一个经典的 k 路网格分割算法。本次实践复习了我许久不用的 C++，锻炼了编写传统算法的能力；另一方面我通过实现各个算法模块，加深了对数据结构课上学到的算法的理解，认识到它们与计算机图形学的联；此外根据图形学课上学到的知识调节超参数实现更好的分割效果，根据各模块运行时间分析瓶颈并加以改进，强化了我的工程应用能力，收获颇丰。

参考文献

[1] S. Katz and A. Tal, “Hierarchical mesh decomposition using fuzzy clustering and cuts,” *ACM transactions on graphics (TOG)*, vol. 22, no. 3, pp. 954–961, 2003. 1