Laplacian Surface Editing 实验报告

121090003 Bao Jingzhi

CUHK(SZ)

日期：2022 年 7 月 18 日

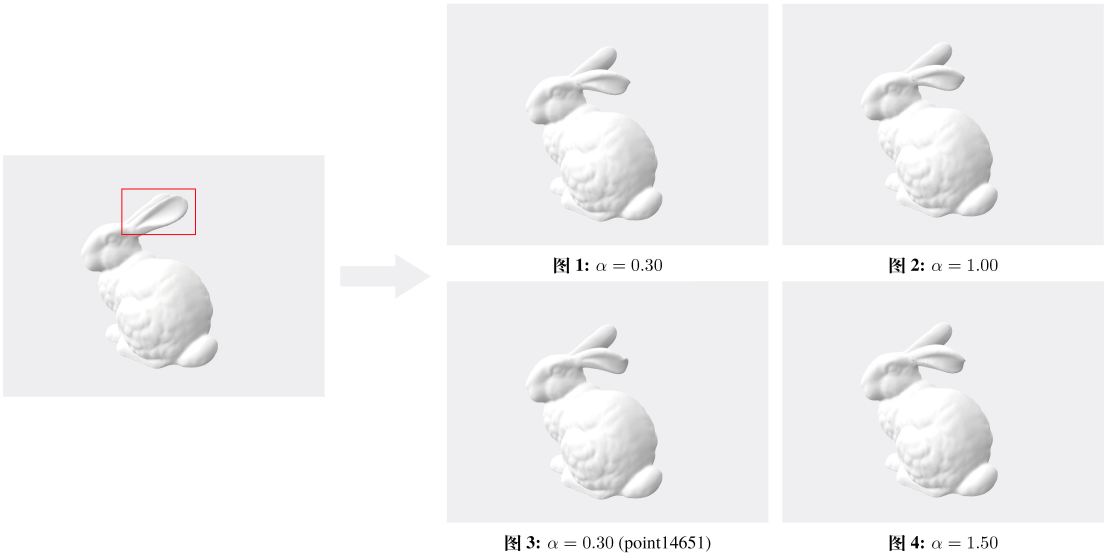


图 1: $\alpha = 0.30$ **图 2:** $\alpha = 1.00$ **图 3:** $\alpha = 0.30$ (point14651) **图 4:** $\alpha = 1.50$

1 预备知识

* 3D 文件 (*.obj, *.mtl, *.ply) 的读写和转换
* 使用 igraph, NetworkX 等工具进行建图
* 利用 Laplacian 算子进行微分坐标与笛卡尔坐标的转换
* 带限制凸优化问题的转换
* Laplacian Surface Editing[1]

2 开发环境

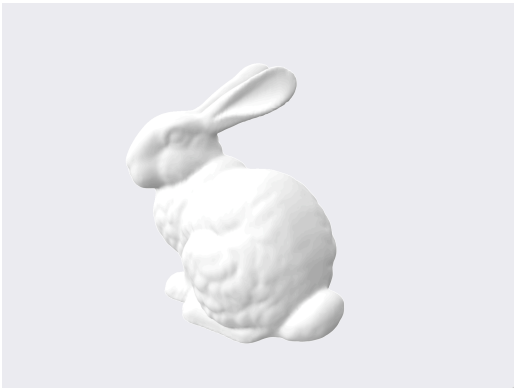
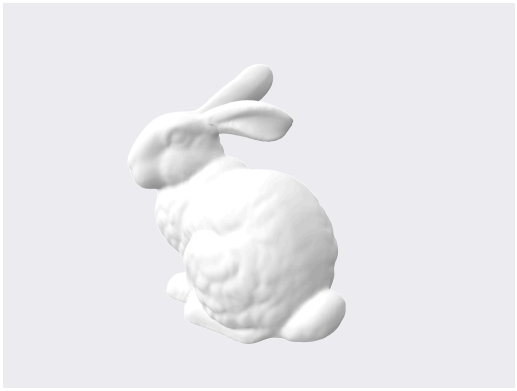
IDE：Microsoft Visual Studio Code (Universal)

Python：3.10.4

[1] Source code is available on GitHub: ZqlwMatt/CV-project.

3 Laplacian Surface Editing

3.1 问题描述

我们需要对指定的 3D 模型进行局部编辑，要求尽可能地保持模型原有的特征。示例图片展示了兔子耳朵的弯曲过程。

3.2 算法描述

3.2.1 思路

对于模型的局部形变操作，顶点的笛卡尔坐标并不重要。这是因为模型的表面特征与**微分坐标**（Laplacian coordinates）相关，因此 Laplacian Surface Editing 的思路是保持模型形变前后的微分坐标不变，**在不改变面结构（拓扑结构）的条件下**保证模型表面的细节特征。

更具体地，微分坐标有 uniform, weighted least squares 和 shape-preserving [Floater 1995] 三种类型，它们都是笛卡尔坐标的凸结合，在 Laplacian Surface Editing 中都适用，论文选用的是 uniform 类型：

$$\delta_i = \mathbf{v}_i - \frac{1}{d_i}\sum_{j\in N_i}\mathbf{v}_j \tag{1}$$

依据上式构造 Laplacian 算子 $L = I - D^{-1}A = D^{-1}L_G$ （L_G 为图的 Laplacian Matrix），微分坐标 Δ 和 V 有转换关系：

$$\Delta = LV \tag{2}$$

设顶点的个数为 n，注意到 $\sum \delta_i = 0$，所以 $\text{Rank}(L) = n - 1$，因此只要在 L 上扩展一维形变点 (handle point) 的信息，根据初始模型的 δ 建立等式 $\delta = L'V$，就可以求出形变后的点的坐标 V。

值得注意的是，我们需要的是**局部区域**经过**旋转**得到的结果，上述方程仅支持全局模型重建。论文的重点是下面的优化方法。

3.2.2 局部编辑

为实现局部编辑，一个想法是固定模型的部分点，仅对部分点作用 Laplacian 算子。这时 $\delta = L'V$ 不一定存在解，问题可以近似转化为最小化函数

$$E(V') = \sum_{i=1}^{n} \|\delta_i - L(\mathbf{v}_i')\|^2 \text{ with } \mathbf{v}_j' = \mathbf{u}_j,\ j \in \{m, \ldots, n\}, m < n. \tag{3}$$

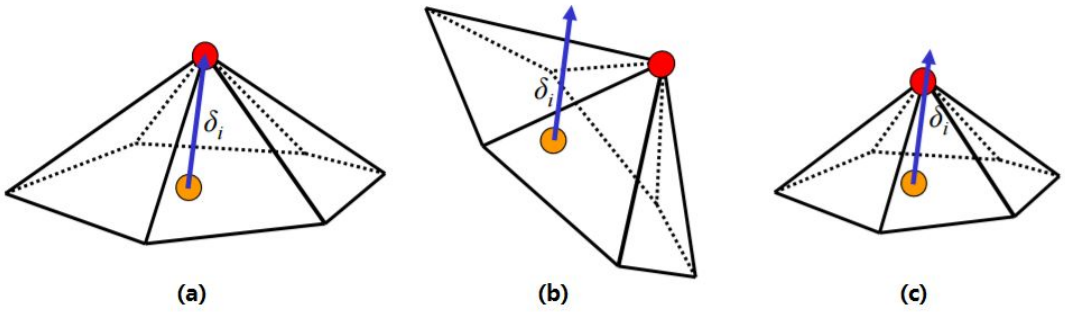
带限制的最优化问题可以转化为

$$E(V') = \sum_{i=1}^{n} \|\delta_i - L(\mathbf{v}_i')\|^2 + \alpha \sum_{i=m}^{n} \|\mathbf{v}_i' - \mathbf{u}_i\|^2. \tag{4}$$

其中 α 为常数，论文给出的函数默认 $\alpha = 1$。

3.2.3 模型仿射变换

微分坐标只保证了模型在平移和全局旋转下的特征，模型 (a) 经过局部变换后得到的模型 (b), (c)，δ_i 不再表示红色点和黄色点形成的向量，但我们认为下图的 (a), (b), (c) 具有相同的细节，因为 (b), (c) 是 (a) 分别经过旋转和缩放得到的。



为了给 δ_i 同时提供旋转和缩放的变换，我们可以引入仿射变换矩阵：

$$T_i = \begin{pmatrix} s & -h_3 & h_2 & t_x \\ h_3 & s & -h_1 & t_y \\ -h_2 & h_1 & s & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

那么问题最终转化为：

$$E(V') = \sum_{i=1}^{n} \|T_i\delta_i - L(\mathbf{v}_i')\|^2 + \alpha \sum_{i=m}^{n} \|\mathbf{v}_i' - \mathbf{u}_i\|^2, \tag{6}$$

$$T_i = \arg\min \left(\left\|T_i\mathbf{v}_i - \mathbf{v}_i'\right\|^2 + \sum_{j \in N_i} \left\|T_i\mathbf{v}_j - \mathbf{v}_j'\right\|^2 \right). \tag{7}$$

3.3 方程求解

求解 Eq.(6) 和 Eq.(7) 是问题的核心，但两者是互相依赖的，无法分别求解。我们显式地写出 T_i 对 V_i 的线性变换，记 $(s_i, \vec{h_i}, \vec{t_i})^T = (s_i, h_{i1}, h_{i2}, h_{i3}, t_{i1}, t_{i2}, t_{i3})^T$。我们希望变换后的点和理想点尽可能地相等，即最小化函数

$$\left\| A_i \left(s_i, \mathbf{h}_i, \mathbf{t}_i\right)^T - \mathbf{b}_i \right\|^2. \tag{8}$$

其中，A_i 为经过调整的邻接矩阵，$\boldsymbol{b}_i$ 为最终点的笛卡尔坐标，更具体地

$$A_i = \begin{pmatrix} v_{k_x} & 0 & v_{k_z} & -v_{k_y} & 1 & 0 & 0 \\ v_{k_y} & -v_{k_z} & 0 & v_{k_x} & 0 & 1 & 0 \\ v_{k_z} & v_{k_y} & -v_{k_x} & 0 & 0 & 0 & 1 \\ \vdots & & & & & & \end{pmatrix}, k \in \{i\} \cup N_i, \tag{9}$$

$$\mathbf{b_i} = \begin{pmatrix} v'_{k_x} \\ v'_{k_y} \\ v'_{k_z} \\ \vdots \end{pmatrix}, k \in \{i\} \cup N_i. \tag{10}$$

这个问题的最小二乘解为

$$(s_i, \vec{h_i}, \vec{t_i})^T = \left(A_i^T A_i\right)^{-1} A_i^T \mathbf{b}_i \tag{11}$$

注意到 T_i 是一个 $\mathbf{b}_i$ 的线性组合，结合 Eq.(11)，可以知道 Eq.(6) 的左半部分 $T_i\delta_i - L(\mathbf{v}'_i)$ 在 L' 中体现为 $\left(A_i^T A_i\right)^{-1} A_i^T - L_i$。对于 L' 的扩展行，则需要根据 Eq.(6) 的右半部分 $\mathbf{v}'_i - \mathbf{u}_i$ ，将 L' 的第 i 行赋值为 α。至此，L' 已经完全构建。

构造矩阵 B

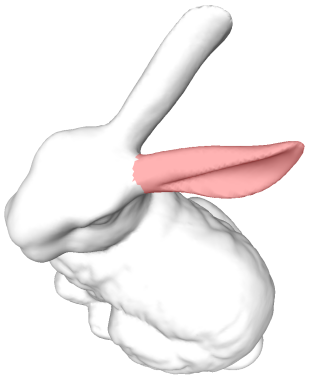
$$B = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \alpha\mathbf{u}_i \\ \alpha\mathbf{u}_{i+1} \\ \vdots \end{pmatrix} \tag{12}$$

Eq.(6) 转化为 $L'V' = B$。方程的最小二乘解就是我们想要的答案。

注：为方便书写，在提供的代码中 Eq.(6) L' 的行按照 x, y, z 的坐标顺序排列，Eq.(7) 按照上文所述的坐标 $v_i(x_i, y_i, z_i)$ 顺序排列。

4 程序实现

4.1 提取局部模型



Laplacian Surface Editing 需要对模型进行局部编辑，我们需要根据提供的固定点求出局部模型，即单连通区域。

```
def get_subgraph(g, ROI_vertices):
    border = []
    for i in range(len(ROI_vertices)):
        segment = g.get_shortest_paths(ROI_vertices[i], ROI_vertices[(i+1)%n],
            output='vpath')[0]
        border += segment[0:-1]
    g1 = g.to_networkx().remove_nodes_from(border)
    return list(nx.node_connected_component(g1, vertex))
```

4.2 Laplacian 算子

```
D = np.diag(sub_g.degree(list(range(n))))
Lap = np.array(sub_g.laplacian())
L = np.linalg.inv(D).dot(Lap)
Delta = np.dot(L, sub_V)
```

4.3 方程求解

```
# Ti x delta
Ti_delta = np.array([
      Delta_ix*   si - Delta_iy*hi[2] + Delta_iz*hi[1], # x 的T变换方程
      Delta_ix*hi[2] + Delta_iy*   si - Delta_iz*hi[0], # y 的T变换方程
    - Delta_ix*hi[1] + Delta_iy*hi[0] + Delta_iz*   si  # z 的T变换方程
])
# Least squares method
Lp = scipy.sparse.coo_matrix(Lp)
Vp = scipy.sparse.linalg.lsqr(Lp, b)[0]
```

5 成果展示

图 1: $\alpha = 0.30$

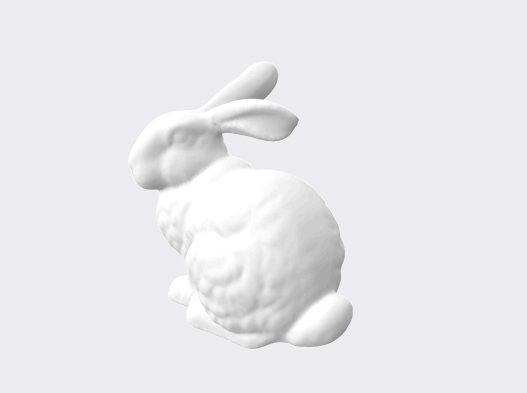


图 2: $\alpha = 1.00$

图 3: $\alpha = 0.30$ (point14651)

图 4: $\alpha = 1.50$

6 算法延伸

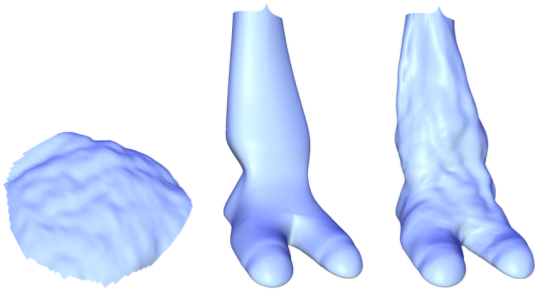
论文中提到了算法的一种应用——涂层迁移，即将一个模型表面的纹理细节映射到另一个模型上。模型表面的涂层可以理解为高频细节 ξ，具体定义为原模型的微分坐标 δ 和经过 Laplacian filter 处理后的模型的微分坐标 $\tilde{\delta}$ 的差值

$$\xi_i = \delta_i - \tilde{\delta}_i \tag{13}$$

现在我们得到了模型 1 的局部高频细节 ξ'，将涂层迁移到模型 2 上只需要将高频细节加到模型 2 的微分坐标 Δ，得到的 U' 即为涂层迁移后的点的笛卡尔坐标

$$U' = L^{-1}\left(\Delta + \xi'\right) \tag{14}$$

但要注意的是，如果我们对不同方向的表面进行涂层迁移，会造成纹理失真，这是因为微分坐标是向量。所以在处理高频细节的时候，我们需要对原模型平滑处理后的表面计算出合适的旋转矩阵。更具体地，点 $\mathbf{v}_i$ 的旋转矩阵可以用其邻接点连边在其切平面上的投影向量表示。

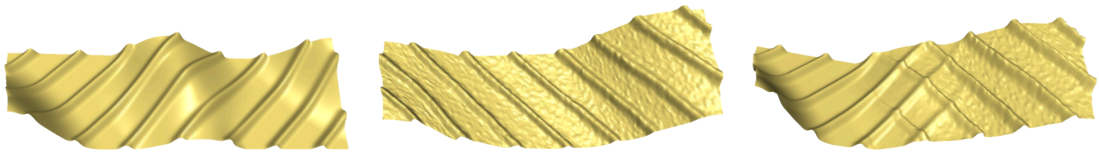


(a) (b) (c)

图 5: Coating transfer

进行局部涂层迁移时还会遇到的一个问题是如何处理模型过度区域的纹理细节。论文采用的方法是混合两者的高频细节，即取两个模型的高频细节的加权均值，这使得模型表面纹理过度更自然。

$$U' = L^{-1}\left(\Delta + \omega_1\xi' + \omega_2\xi_0\right),\ w_1 + w_2 = 1 \tag{15}$$



(a) (b) (c)

图 6: Texture mapping

7 总结

Laplacian Surface Editing 和 Poisson Image Editing 的思想如出一辙，都是通过保持离散点的 Laplacian Coordinates 进行模型/图像重建，从而保证细节完整。特别地，两者的涂层迁移算法也具有相似性，都是通过取加权和的方式使得过渡区表现自然。

Laplacian Surface Editing 的亮点是构造了旋转矩阵，在对 Laplacian Coordinates 进行方向旋转的同时，使方程仍为线性。

但这种方法的缺点也很明显，在全局重建过程中，如果模型较大，编辑区域顶点产生的矩阵会平方倍增大，内存骤增。方程求解和编辑区域重新参数化的时间也会骤增。

参考文献

[1] Michael S Floater. “Parametrization and smooth approximation of surface triangulations”. In: *Computer aided geometric design* 14.3 (1997), pp. 231–250.

[2] Olga Sorkine et al. “Laplacian surface editing”. In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 2004, pp. 175–184.