



GAMES 301: 曲面参数化 作业报告

GAMES 301: Surface Parameterization Homework Report

作者: Mason Wu

组织: Infinite Heaven

版本: 0.02

Mason Wu: Simplicity ≠ Mediocrity.



ElegantLATEX Program

目录

第 1 章 Least Squares Conformal Maps and Mean Value Coordinates	1
1.1 摘要	1
1.2 引言	1
1.3 符号表示	1
1.4 预备知识	1
1.4.1 Tutte's Embedding	2
1.5 基于最小二乘共形映射的自由边界无翻转参数化方法	3
1.5.1 共形映射	3
1.5.2 最小二乘共形映射 (LSCM)	4
1.5.3 Mean Value Coordinates	5
1.6 实验结果	5
1.7 总结	7
附录 A 代码修改说明	8
A.1 对原有文件的修改	8
A.2 新增文件	9

第 1 章 Least Squares Conformal Maps and Mean Value Coordinates

1.1 摘要

曲面参数化技术在模型贴图、曲面纹理展开等方向有重要应用。我们在作业 1 中使用固定边界的 Tutte's Embedding 完成了基于 Tutte 权重 [1, 2] 和 Floater 权重 [3] 的三维三角形曲面的平面嵌入。然而，我们固定的边界形状为凸多边形，导致具有非凸边界的三角形曲面的平面参数化结果存在较大的扭曲。本次报告，我们将实现 Lévy [4] 等人提出的最小二乘共形映射曲面参数化方法 (Least Squares Conformal Maps, LSCM)，并在求得边界后使用基于 Mean Value Coordinates 权重 [5] 和 Tutte's Embedding 方法生成无自交的、无翻转的、满足共形映射的自由边界平面参数化结果。

1.2 引言

Tutte's Embedding 适用于凸的固定边界的平面参数化，Tutte 证明了 [2] 在边界点为凸边界，且每个内部点是邻居节点的凸组合的情况下，Tutte's Embedding 将得到具有凸面的平面嵌入结果。Floater [6, 7] 将 Tutte 的“凸边界”条件推广至非严格凸（相邻的边界顶点可以贡献）的情形，并且当内部顶点被放置在邻居顶点坐标的广义凸组合的位置上时成立。Gortler [8] 等人提到，在许多场景下，从 3D 到 2D 的共形映射可以保持夹角，也叫“保角映射”。基于 Floater [5] 的 mean-value 权重总是可以取得正值，可用于得到接近共形映射的参数化结果。强制所有角度在区间 $[0, \pi]$ 内取值（附带其他约束条件）保证了所得到的 2D 平面三角化结果是单射的。这种方法是非线性的，但具有自由边界的优势。本报告主要记录实现 LSCM 和 Mean Value Coordinates 权重过程中遇到的问题和解决方案。

1.3 符号表示

表 1.1：文中所使用的符号及涵义

符号	涵义
\mathbb{R}	数值空间
x	标量，实数
U	复数，表示为 $U = u + iv$
\mathbf{x}	向量，元素可以是实数也可以是复数，取决于上下文
\mathbf{X}	集合，指网格模型的顶点位置的径向集合 $\mathbf{X} = \{\mathbf{x}_i = (x_i, y_i, z_i)\}_{i=1}^N$
\mathbf{X}	矩阵， $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m]$

1.4 预备知识

我们部分采用文章 [9] 中的符号，令 $\bar{\mathbf{x}}$ 表示输入三角网格 $S(G, \mathbf{X})$ 的顶点位置， \mathbf{x} 表示优化过程中的平面参数化结果 $\mathcal{P} = \mathcal{P}(G, U_b)$ 中的顶点位置 ($U_b = \{P_1, P_2, \dots, P_n\}$)。

1.4.1 Tutte's Embedding

在作业 1 中我们实现了 Tutte's Embedding 的相关内容。在这进行简要的介绍。

曲面参数化方法是找到从三维曲面到二维区域的映射。对于三角形曲面，参数化方法是找到映射：

$$f : \mathbf{x}_i = (x_i, y_i, z_i)^T \mapsto P_i = (u_i, v_i).$$

对于 $x_i \in V$, $i \in \{n+1, n+2, \dots, N\}$, 选择 $\lambda_{i,j}$ 满足

$$\begin{cases} \lambda_{i,j} = 0, & (i,j) \notin E, \\ \lambda_{i,j} > 0, & (i,j) \in E, \end{cases} \quad \text{且} \quad \sum_{j=1}^N \lambda_{i,j} = 1. \quad (1.1)$$

定义 $P_{n+1}, P_{n+2}, \dots, P_N$ 是线性方程组的解：

$$P_i = \sum_{j=1}^N \lambda_{i,j} P_j, \quad i = n+1, n+2, \dots, N. \quad (1.2)$$

令 $\mathcal{P} = \mathcal{P}(G, U_b, \Lambda)$, $U_b = \{P_1, P_2, \dots, P_n\}$, 并且 $\Lambda = (\lambda_{i,j})_{i=n+1, \dots, N, j=1, \dots, N}$ 作为 $G(V, E, F)$ 在 \mathbb{R}^2 的顶点 $P_{n+1}, P_{n+2}, \dots, P_N$ 中的嵌入, 且满足边是直线, 面是三角形。

定理 1.1 (Floater [3])

令 G_i 是 G 中包含结点 x_i 及其所有邻居结点和邻接关系的子图, $\mathbf{x}_{j_1}, \mathbf{x}_{j_2}, \dots, \mathbf{x}_{j_{d_i}}$ 是满足逆时针顺序的邻接结点, 那么在公式 1.1 的选取方式下, 矩阵

$$\mathbf{A} = \begin{bmatrix} j & 1 & \cdots & n & n+1 & \cdots & N \\ P_1 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ P_n & 0 & \cdots & 1 & 0 & \cdots & 0 \\ P_{n+1} & \cdots & -\lambda_{n+1,j_p} & \cdots & 1 & -\lambda_{n+1,j_q} & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots \\ P_N & \cdots & -\lambda_{N,j_p} & \cdots & -\lambda_{N,j_q} & \cdots & 1 \end{bmatrix} \quad (1.3)$$

是非奇异的。



于是根据矩阵 1.2 及方程 1.3 可以得到我们要求解的方程 $Ax = b$, 即

$$\begin{bmatrix} j & 1 & \cdots & n & n+1 & \cdots & N \\ P_1 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ P_n & 0 & \cdots & 1 & 0 & \cdots & 0 \\ P_{n+1} & \cdots & -\lambda_{n+1,j_p} & \cdots & 1 & -\lambda_{n+1,j_q} & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots \\ P_N & \cdots & -\lambda_{N,j_p} & \cdots & -\lambda_{N,j_q} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u & v \\ x_0 & y_0 \\ \vdots & \vdots \\ x_n & y_n \\ x_{n+1} & y_{n+1} \\ \vdots & \vdots \\ x_N & y_N \end{bmatrix} = \begin{bmatrix} x & y \\ P_1.x & P_1.y \\ \vdots & \vdots \\ P_n.x & P_n.y \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix}. \quad (1.4)$$

其中 P_1, \dots, P_n 是边界点在 \mathbb{R}^2 上的坐标, 本报告中 $P_i \in [0, 1]^2$, $i \in \{1, 2, \dots, n\}$ 。

则算法流程如 1:

Algorithm 1: Tutte Embedding 算法流程

Data: 三角化曲面 $S(G, \mathbf{X})$, $G = G(\mathbf{V}, \mathbf{E}, \mathbf{F})$, $\mathbf{X} = \{\mathbf{x}_i = (x_i, y_i, z_i)\}_{i=1}^N$

Output: $\mathcal{P} = \mathcal{P}(G, U_b, \Lambda)$, $U_b = \{P_1, P_2, \dots, P_n\}$ 并且 $\Lambda = (\lambda_{i,j})_{i=n+1, \dots, N, j=1, \dots, N}$

- 1 计算 S 的边界 ∂G , 根据结果对结点重新编号;
- 2 将正 n 边形的顶点坐标赋予 $b_{N \times 2} = [P_1, P_2, \dots, P_n, 0, \dots, 0]^T$;
- 3 **for** $k = n + 1$ **to** N **do**
- 4 | 基于 Tutte 或者 Floater 的方法计算 λ_{k,j_l} , $l \in \{1, 2, \dots, d_k\}$;
- 5 **end**
- 6 根据公式 1.3 计算 A 并求解方程 1.4 得到 $x_{N \times 2} = [P_1, P_2, \dots, P_n, P_{n+1}, \dots, P_N]^T$;
- 7 输出 $\mathcal{P} = \mathcal{P}(G, U_b, \Lambda)$;

1.5 基于最小二乘共形映射的自由边界无翻转参数化方法

1.5.1 共形映射

如右图 1.1 所示, 从 (u, v) 域到三角形曲面的共形映射满足 u -等值线和 v -等值线在过点 \mathcal{X} 处的切向量正交, 且在曲面上有相同的法向量, 即

$$N(u, v) \times \frac{\partial \mathcal{X}}{\partial u}(u, v) = \frac{\partial \mathcal{X}}{\partial v}(u, v). \quad (1.5)$$

其中 $N(u, v)$ 表示曲面上的单位法向量。

注 共形映射是局部各向异性的, 将 (u, v) 域的基本圆映射到曲面上的基本圆。

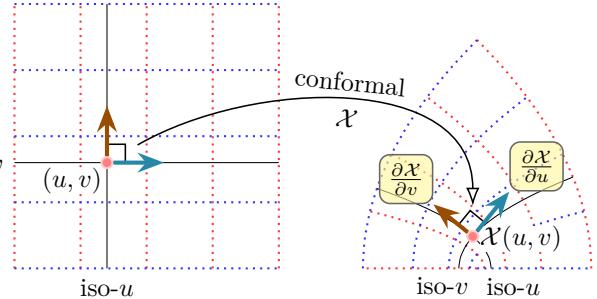


图 1.1: 共形映射

我们对三角化曲面 $S(G, \mathbf{X})$ 的每个三角形 T 建立局部坐标系如右图 1.2 所示。假定三角形的三个顶点由局部坐标系给出, 则三个顶点的坐标分别为 (x_1, y_1) , (x_2, y_2) 和 (x_3, y_3) 。考虑图 1.1 中对于三角形 T 上共形映射对点 \mathcal{X} 的限制, 并将共形性应用到逆映射 $\mathcal{U} : (x, y) \mapsto (u, v)$, 即求给定点坐标的参数化位置。在三角形内, 公式 1.5 为

$$\frac{\partial \mathcal{X}}{\partial u} - i \frac{\partial \mathcal{X}}{\partial v} = 0, \quad (1.6)$$

其中 \mathcal{X} 为复数形式 $\mathcal{X} = x + iy$ 。考虑逆映射 \mathcal{U} , 则有

$$\frac{\partial \mathcal{U}}{\partial x} + i \frac{\partial \mathcal{U}}{\partial y} = 0, \quad (1.7)$$

成立, 其中 $\mathcal{U} = u + iv$, 说明逆映射 \mathcal{U} 满足柯西-黎曼公式。

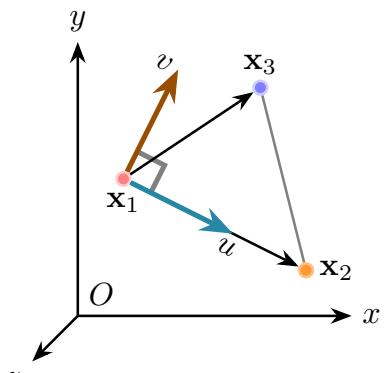


图 1.2: 三角形局部坐标

三角形网格不能严格满足公式 1.7, 我们考虑尽量减少违反公式的最小二乘问题, 并定义指标 C :

$$C(T) = \int_T \left| \frac{\partial \mathcal{U}}{\partial x} + i \frac{\partial \mathcal{U}}{\partial y} \right|^2 dA = \left| \frac{\partial \mathcal{U}}{\partial x} + i \frac{\partial \mathcal{U}}{\partial y} \right|^2 A_T, \quad (1.8)$$

其中 A_T 是三角形的(定向)面积, 记号 $|z|$ 表示复数 z 的模。将所有三角形的指标求和, 即可得到需要最小化的目标函数

$$C(\mathbf{T}) = \sum_{T \in \mathbf{T}} C(T). \quad (1.9)$$

1.5.1.1 三角形内的梯度

为了给每个三角形的顶点 v_j 赋予复数 U_j 使其满足柯西黎曼方程（最小二乘意义上），我们需要将 $C(T)$ 重写，并假定映射 \mathcal{U} 在 T 内线性变化。考虑 \mathbb{R}^2 中三角形 T 内的顶点坐标 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ 和关联的常量 u_1, u_2, u_3 ，有

$$\begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} = \frac{1}{d_T} \begin{pmatrix} y_2 - y_3 & y_3 - y_1 & y_1 - y_2 \\ x_3 - x_2 & x_1 - x_3 & x_2 - x_1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

其中 $d_T = (x_1y_2 - y_1x_2) + (x_2y_3 - y_2x_3) + (x_3y_1 - y_3x_1)$ 是三角形面积的两倍。

梯度的两个成分 $\partial u / \partial x$ 和 $\partial u / \partial y$ 可以组合为一个复数

$$\frac{\partial u}{\partial x} + i \frac{\partial u}{\partial y} = \frac{i}{d_T} (W_1 \quad W_2 \quad W_3) (u_1 \quad u_2 \quad u_3)^\top,$$

其中

$$\begin{cases} W_1 = (x_3 - x_2) + i(y_3 - y_2), \\ W_2 = (x_1 - x_3) + i(y_1 - y_3), \\ W_3 = (x_2 - x_1) + i(y_2 - y_1). \end{cases}$$

公式 1.7 可以写作

$$\frac{\mathcal{U}}{\partial x} + i \frac{\mathcal{U}}{\partial y} = \frac{i}{d_T} (W_1 \quad W_2 \quad W_3) (U_1 \quad U_2 \quad U_3)^\top,$$

其中 $U_j = u_j + iv_j$ 。因此目标函数可以写为

$$C(\mathbf{U} = (U_1, \dots, U_n)^\top) = \sum_{T \in \mathcal{T}} C(T),$$

而

$$C(T) = \frac{1}{d_T} \left| (W_{j_1, T} \quad W_{j_2, T} \quad W_{j_3, T}) (U_{j_1} \quad U_{j_2} \quad U_{j_3})^\top \right|,$$

三角形 T 的顶点下标为 j_1, j_2 和 j_3 。

1.5.2 最小二乘共形映射 (LSCM)

上一节的 $C(\mathbf{U})$ 是关于复数 U_1, \dots, U_n 的二次形式，可以写为

$$C(\mathbf{U}) = \mathbf{U}^* \mathcal{C} \mathbf{U}, \tag{1.10}$$

其中 \mathcal{C} 是 $|\mathcal{V}| \times |\mathcal{V}|$ 的对称厄密特矩阵， \mathbf{U}^* 表示 \mathbf{U} 的厄密特（复）共轭。 \mathcal{C} 是厄密特格拉姆矩阵，可以写为

$$\mathcal{C} = \mathcal{M}^* \mathcal{M},$$

其中 $\mathcal{M} = (m_{ij})$ 是稀疏的 $|\mathcal{F}| \times |\mathcal{V}|$ 的矩阵，行对应三角面，列对应顶点。 \mathcal{M} 的系数为

$$m_{ij} = \begin{cases} \frac{W_{j,T_i}}{\sqrt{d_{T_i}}} & \text{如果顶点 } v_j \text{ 属于三角形 } T_i, \\ 0 & \text{其它.} \end{cases}$$

为了得到最优化问题的非平凡解，我们需要预先指定部分 U_i 的取值。向量 \mathbf{U} 可以分解为 $(\mathbf{U}_f^\top, \mathbf{U}_p^\top)^\top$ ，其中 \mathbf{U}_f 是向量 \mathbf{U} 中的自由坐标，也是优化问题中的变量。 \mathbf{U}_p 是向量 \mathbf{U} 中的固定坐标，长度为 $p \leq |\mathcal{V}|$ 。

类似的， \mathcal{M} 也可以被分解为分块矩阵

$$\mathcal{M} = \begin{pmatrix} \mathcal{M}_f & | & \mathcal{M}_p \end{pmatrix},$$

其中 \mathcal{M}_f 是大小为 $|F| \times (|V| - p)$ 的矩阵， \mathcal{M}_p 是大小为 $|F| \times p$ 的矩阵。于是公式 1.10 可被写为

$$C(\mathbf{U}) = \mathbf{U}^* \mathcal{M}^* \mathcal{M} \mathbf{U} = \|\mathcal{M} \mathbf{U}\|^2 = \|\mathcal{M}_f \mathbf{U}_f + \mathcal{M}_p \mathbf{U}_p\|^2,$$

其中 $\|\mathbf{v}\|$ 表示内积 $\langle \mathbf{v}, \bar{\mathbf{v}} \rangle$ ($\bar{\mathbf{v}}$ 表示 \mathbf{v} 的共轭)。

于是，我们可以将目。记 $\mathcal{M} = \mathcal{M}^{\text{Re}} + i\mathcal{M}^{\text{Im}}$ ， \mathcal{M}^{Re} 和 \mathcal{M}^{Im} 都是实矩阵，并将求解目标函数的最小二乘问题改写为实矩阵方程的形式：

$$C(\mathbf{x}) = \|\mathcal{A}\mathbf{x} - \mathbf{b}\|^2, \quad (1.11)$$

且

$$\mathcal{A} = \begin{pmatrix} \mathcal{M}_f^{\text{Re}} & -\mathcal{M}_f^{\text{Im}} \\ \mathcal{M}_f^{\text{Im}} & -\mathcal{M}_f^{\text{Re}} \end{pmatrix}, \quad \mathbf{b} = - \begin{pmatrix} \mathcal{M}_p^{\text{Re}} & -\mathcal{M}_p^{\text{Im}} \\ \mathcal{M}_p^{\text{Im}} & -\mathcal{M}_p^{\text{Re}} \end{pmatrix} \begin{pmatrix} \mathbf{U}_p^{\text{Re}} \\ \mathbf{U}_p^{\text{Im}} \end{pmatrix},$$

上标 Re 和 Im 分别代表了矩阵的实部和虚部。这次记号 $\|\mathbf{v}\|$ 表示实向量的 L_2 -范数， $\mathbf{x} = (\mathbf{U}_f^{\text{Re}\top}, \mathbf{U}_f^{\text{Im}\top})^\top$ 是待求向量。注意实矩阵 \mathcal{A} 的大小为 $2|F| \times 2(|V| - p)$ ， $\mathbf{b} \in \mathbb{R}^{2|F|}$ ， $\mathbf{x} \in \mathbb{R}^{2(|V|-p)}$ 。

文章 [4] 中证明了以下几点：

- 当固定点的个数大于等于 2 时，矩阵 \mathcal{A} 是满秩的；
- 当 $p \geq 2$ 时，最小化问题有唯一解 $\mathbf{x} = (\mathcal{A}^\top \mathcal{A})^{-1} \mathcal{A}^\top \mathbf{b}$ 。 p 的最优值为 2，因为在此情况下，如果曲面可展（目标函数最小值为 0），则映射 \mathcal{U} 是完全共形的。
- 最小化问题的解不随纹理空间相似度改变；
- 最小化问题的与三角形网格分辨率不相关；
- 纹理空间中，如果选择固定 F 中的边界点，则所有的三角形是一致定向的，不会发生翻转。

1.5.3 Mean Value Coordinates

Floater [5] 提出了即便邻居节点依序组成非凸多边形 (Star-shaped)，也可以得到邻居的“正权重”的计算方法。根据 Gortler [8] 等人的工作，可知对于边界顶点在是其邻域凸包内，且内部顶点是其邻居节点的凸组合时，可得到无自交的平面参数化结果。对于节点 v_0 ，其邻居节点是 v_1, \dots, v_k 。目的是求满足

$$\sum_{i=1}^k \lambda_i v_i = v_0, \quad \sum_{i=1}^k \lambda_i = 1.$$

的广义重心坐标 $\lambda_1, \dots, \lambda_k$ 。如右图，对应的坐标形式为

$$\lambda_i = \frac{w_i}{\sum_{j=1}^k w_j}, \quad w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|v_i - v_0\|} = \frac{1}{r_i} \left(\tan \frac{\alpha_{i-1}}{2} + \tan \frac{\alpha_i}{2} \right). \quad (1.12)$$

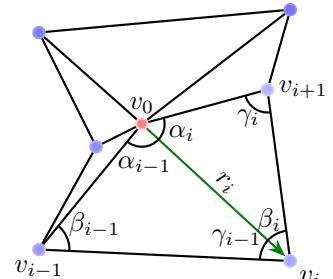


图 1.3: Star-shaped 多边形

1.6 实验结果

我们使用五个模型测试本文方法的有效性 (表 1.2)，固定边界点的规则为：选取边界点中的第一个顶点，以及位于序列中间顺序的顶点作为固定点，固定位置不限。

名称	格式	面类型	V	E	F	边界数	最长边界长度	存储
cathead	OBJ	三角面	131	378	248	1	12	8KB
Balls	OBJ	三角面	547	1578	1032	1	60	26KB
bunnyhead	OBJ	三角面	741	2188	1448	1	32	55KB
hand	OFF	三角面	1558	4653	3096	1	18	97KB
cow	OBJ	三角面	3195	8998	5804	1	584	194KB

表 1.2: 测试数据

实验结果如图 1.4, 观察发现共形映射的“将参数域的基本圆映射到曲面上的基本圆”现象。共形映射很好地保持了 UV 贴图上的圆形形状, 但部分区域仍存在巨大的扭曲。

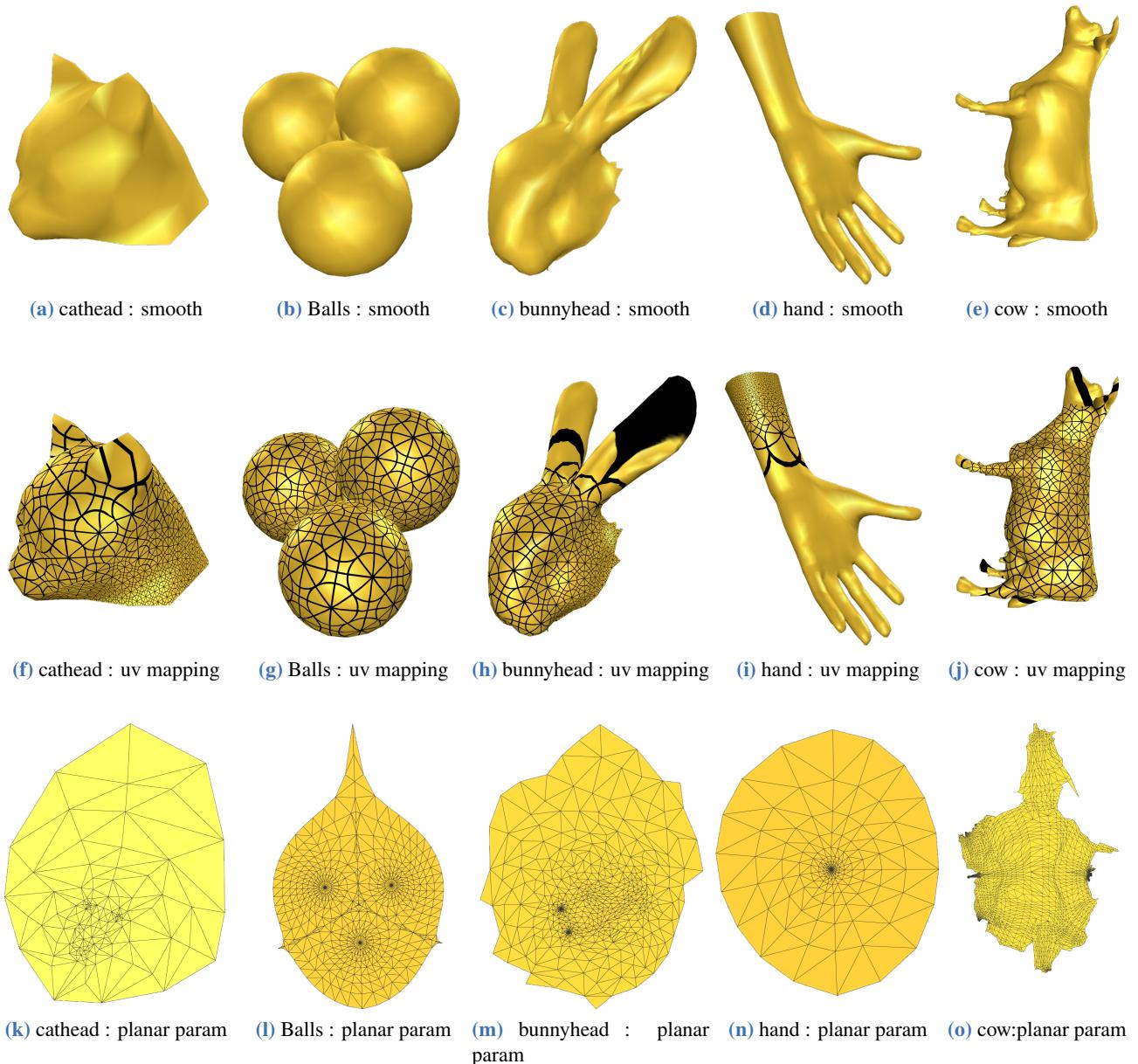


图 1.4: LSCM 和 Mean-value Coordinates 的 Tutte's Embedding 结果。第一排为数据的平滑着色结果, 第二排为采用纹理贴图展示基本圆的映射结果, 第三排为共形映射平面参数化结果。

方法用时如表 1.3。

名称	V	E	F	LSCM 用时 (ms)	Tutte's Embedding 用时 (ms)	总用时 (ms)
cathead	131	378	248	0.606	0.457	1.596
Balls	547	1578	1032	6.714	1.838	9.452
bunnyhead	741	2188	1448	9.454	2.537	13.277
hand	1558	4653	3096	24.617	6.039	33.280
cow	3195	8998	5804	320.225	15.599	344.261

表 1.3: 方法用时

注意表 1.3 中的总用时略大于 LSCM 用时和 Tutte's Embedding 用时，因为总用时额外统计了解法器加载模型，以及写回纹理坐标的时间。可以看出运行时间还是随着模型的规模显著增加的。

1.7 总结

本次作业实现了 [4] 提出的最小二乘共形映射方法，以及 Tutte's Embedding 框架下的 Mean-value Coordinates 权重计算方法，虽然对于某些数据，因为固定点的选择不合适，导致结果存在局部自交，但是总体而言，LSCM 是一个优秀的方法。不仅计算速度快，同时能得到不错的自由边界参数化结果。

附录 A 代码修改说明

A.1 对原有文件的修改

Listing A.1: surfacemeshprocessing.*

```
1 // ===== surfacemeshprocessing.h =====
2 class SurfaceMeshProcessing : public QMainWindow
3 {
4     // ===== hw3: Actions =====
5     QAction* actFreeBoundarySolver;
6     // ...
7 };
8
9 // ===== surfacemeshprocessing.cpp =====
10 void SurfaceMeshProcessing::CreateActions(void)
11 {
12     // ...
13     // ===== hw3: Actions =====
14     actFreeBoundarySolver = new QAction("Free Boundary Solver", this);
15     actFlat->setStatusTip(tr("Conformal Param with Non-convex Boundary"));
16     connect(actFreeBoundarySolver, SIGNAL(triggered()), viewer, SLOT(FreeBoundarySolver()));
17 }
```

Listing A.2: MainViewerWidget.*

```
1 // ===== MainViewerWidget.h =====
2 class MainViewerWidget : public QDialog
3 {
4     // ...
5     // ===== hw3: Free Boundary Solver =====
6     void FreeBoundarySolver();
7     // ...
8 };
9
10 // ===== MainViewerWidget.cpp =====
11 // ...
12 // ===== hw3: Free Boundary Solver API =====
13 void MainViewerWidget::FreeBoundarySolver()
14 {
15     meshviewerwidget->FreeBoundarySolver();
16 }
17 // ...
```

Listing A.3: MeshViewerWidget.*

```

1 // ===== MeshViewerWidget.h =====
2 class MeshViewerWidget : public QGLViewerWidget
3 {
4     // ...
5     // ===== hw3: Boundary Free Method =====
6     void FreeBoundarySolver();
7     // ...
8     // ===== hw3: Free Boundary member =====
9     bool isFreeBoundarySolver = false;
10
11     freeb::FreeBoundarySolver m_FreeBoundarySolver;
12 };
13
14 // ===== MeshViewerWidget.cpp =====
15 // ...
16 // ===== hw3: Free Boundary Solver API =====
17 void MeshViewerWidget::FreeBoundarySolver()
18 {
19     if (polyMesh->numVertices() == 0)
20     {
21         std::cerr << "ERROR: ProjNewtonSolver() No vertices!" << std::endl;
22         return;
23     }
24
25     auto timeStart = std::chrono::steady_clock::now();
26
27     m_FreeBoundarySolver.Solve(polyMesh);
28
29     auto timeEnd = std::chrono::steady_clock::now();
30     auto ms = std::chrono::duration_cast<std::chrono::microseconds>(timeEnd - timeStart).count() / 1000.0;
31
32     std::cout << "[Free Boundary] Solver finished with " << ms << " ms\n";
33     isFreeBoundarySolver = false;
34     update();
35 }
36 // ...

```

A.2 新增文件

文件 在相关代码存放在目录 Surface_Framework_Cmake/src/homeworks/FreeBoundary/:

- (a) Util_FreeBoundary.h
- (b) Util_FreeBoundary.cpp

Listing A.4: Util_FreeBoundary.h

```

1 // ===== Util\_FreeBoundary.h =====
2 #pragma once
3
4 #include <queue>
5 #include <vector>
6 #include <Eigen/Sparse>
7
8 #include " ../PolyMesh/include/PolyMesh/PolyMesh.h"
9
10 namespace freeb
11 {
12     class FreeBoundarySolver
13     {
14     public:
15         void Solve(acamcad::polymesh::PolyMesh* mesh);
16
17     private:
18         void LSCMUV(acamcad::polymesh::PolyMesh* mesh);
19
20         void TutteWithMeanValue(acamcad::polymesh::PolyMesh* mesh);
21         void ConstrainUV(acamcad::polymesh::PolyMesh* mesh) const;
22
23         std::vector<double> boundaryEdgesLength(const std::vector<acamcad::polymesh::MVert*>& boundaries) const;
24         std::vector<double> MeanValueWeight(acamcad::polymesh::MVert* v,
25             const std::vector<acamcad::polymesh::MVert*>& adjVerts) const;
26
27     private:
28         // data member
29         Eigen::VectorXd m_UVList;
30
31         // statistics
32         size_t m_Iters{ 0 };
33     };
34 }
```

Listing A.5: Util_FreeBoundary.cpp

```

1 // ===== Util\_FreeBoundary.cpp =====
2 #include "Util_FreeBoundary.h"
3
4 #include <algorithm>
5 #include <iostream>
6 #include <numeric>
7 #include <chrono>
8
9 #include <Eigen\.Dense>
10 #include <Eigen\Sparse>
11 #include <Eigen\SparseQR>
```

```

12 #include <Eigen\IterativeLinearSolvers>
13
14 namespace freeb
15 {
16     void FreeBoundarySolver::Solve(acamcad::polymesh::PolyMesh* mesh)
17     {
18         m_UVList = Eigen::VectorXd(mesh->vertices().size() * 2);
19         m_UVList.setZero();
20
21         // step 1. get coarse boundary with LSCM
22         LSCMUV(mesh);
23
24         // step 2. calculate tutte's parameterization with mean-value coordinate
25         TutteWithMeanValue(mesh);
26
27         // step 3. write back
28         ConstrainUV(mesh);
29     }
30
31     void FreeBoundarySolver::LSCMUV(acamcad::polymesh::PolyMesh* mesh)
32     {
33         size_t numV = mesh->vertices().size();
34         size_t numF = mesh->polyfaces().size();
35
36         // boundaries
37         auto meshBoundaries = mesh->boundaryVertices();
38         size_t pID0 = meshBoundaries[0]->index();
39         size_t pID1 = meshBoundaries[meshBoundaries.size() / 2]->index();
40         size_t numP = 2;
41
42         std::cout << "[Free Boundary] pinned index = [" << pID0 << ", " << pID1 << "]\n";
43
44         Eigen::Vector2d fixP0;
45         fixP0 << -0.5, 0.0;
46         Eigen::Vector2d fixP1;
47         fixP1 << 0.5, 0.0;
48         m_UVList.segment(pID0 * 2, 2) = fixP0;
49         m_UVList.segment(pID1 * 2, 2) = fixP1;
50
51         std::vector<acamcad::polymesh::MVert*> pinnedVert(numP, nullptr);
52
53         // A = [ Re(Mf) -Im(Mf) ]
54         //      [ Im(Mf) Re(Mf) ](2F * 2(V-P))
55         //      Mf(F * (V-P))
56         Eigen::SparseMatrix<double> matA = Eigen::SparseMatrix<double>(2 * numF, 2 * (numV
57         - numP));
58         // B = [ Re(Mp) -Im(Mp) ]
59         //      [ Im(Mp) Re(Mp) ](2F * 2P)
          //      Mp(F * P)

```

```

60 Eigen::SparseMatrix<double> matB = Eigen::SparseMatrix<double>(2 * numF, 2 * numP);
61
62 std::vector<Eigen::Triplet<double>> matAtriples;
63 std::vector<Eigen::Triplet<double>> matBtriples;
64
65 std::cout << "[Free Boundary] LSCM Start\n";
66 auto timeStart = std::chrono::steady_clock::now();
67
68 std::unordered_map<acamcad::polymesh::MVert*, size_t> freeVertIndexMap;
69 {
70     std::vector<acamcad::polymesh::MVert*> freeVert;
71     freeVert.reserve(numV - numP);
72     for (auto* pV : mesh->vertices())
73     {
74         if (pV->index() == pID0 || pV->index() == pID1) continue;
75         freeVert.emplace_back(pV);
76     }
77
78     for (size_t i = 0; i < numV - numP; ++i) freeVertIndexMap.insert({ freeVert[i], i });
79 }
80
81 // step 1. position on local orthonormal bases
82 for (size_t faceID = 0; faceID < numF; ++faceID)
83 {
84     auto faceVerts = mesh->polygonVertices(mesh->polyface(faceID));
85     auto v21 = faceVerts[1]->position() - faceVerts[0]->position();
86     auto v31 = faceVerts[2]->position() - faceVerts[0]->position();
87     // y
88     // ^
89     // |   v3
90     // .---.
91     // ^   / \ \
92     // |   /   \
93     // |   /   \
94     // |   v   \
95     // .==>--->.-> x
96     // v1   vp   v2
97     Eigen::Vector2d P1{ 0, 0 };
98     Eigen::Vector2d P2{ v21.norm(), 0 };
99     Eigen::Vector2d P3{ v21.dot(v31) / v21.norm(), v21.cross(v31).norm() / v21.norm() };
100
101    double faceArea2 = (P1.x() * P2.y() - P1.y() * P2.x()) +
102        (P2.x() * P3.y() - P2.y() * P3.x()) +
103        (P3.x() * P1.y() - P3.y() * P1.x());
104
105    std::array<Eigen::Vector2d, 3> Ws = { Eigen::Vector2d{ P3.x() - P2.x(), P3.y() -
P2.y() } / std::sqrt(faceArea2),

```

```

106                 Eigen::Vector2d{ P1.x() - P3.x(), P1.y() - P3.y() } / std::
107                 sqrt(faceArea2),
108                 Eigen::Vector2d{ P2.x() - P1.x(), P2.y() - P1.y() } / std::
109                 sqrt(faceArea2) };

110             // step 2. calculate matrix A and matrix B
111             for (size_t vID = 0; vID < 3; ++vID)
112             {
113                 size_t vertID = faceVerts[vID]->index();
114                 auto W = Ws[vID];
115
116                 if (vertID == pID0 || vertID == pID1)
117                 {
118                     size_t targetVertID = (vertID == pID1);
119                     matBtriples.emplace_back(faceID, targetVertID, W.x());
120                     matBtriples.emplace_back(faceID + numF, targetVertID + numP, W.x());
121                     matBtriples.emplace_back(faceID, targetVertID + numP, -W.y());
122                     matBtriples.emplace_back(faceID + numF, targetVertID, W.y());
123                 } else {
124                     size_t targetVertID = freeVertIndexMap[faceVerts[vID]];
125                     matAtriples.emplace_back(faceID, targetVertID, W.x());
126                     matAtriples.emplace_back(faceID + numF, targetVertID + numV - numP, W.x());
127                     matAtriples.emplace_back(faceID, targetVertID + numV - numP, -W.y());
128                     matAtriples.emplace_back(faceID + numF, targetVertID, W.y());
129                 }
130             }

131             matA.setFromTriplets(matAtriples.begin(), matAtriples.end());
132             matA.makeCompressed();
133             matB.setFromTriplets(matBtriples.begin(), matBtriples.end());

134             { // step 3. calculate matrix Mf and Mp
135                 Eigen::Vector4d vecU;
136                 vecU << fixP0.x(), fixP1.x(), fixP0.y(), fixP1.y();
137
138                 // b = -B [ Re(Up) Im(Up) ]T
139                 Eigen::VectorXd vecb = -matB * vecU;
140
141                 Eigen::LeastSquaresConjugateGradient<Eigen::SparseMatrix<double>> solver;
142                 solver.compute(matA);
143                 Eigen::VectorXd X = solver.solve(vecb);

144                 size_t bIndex = 0;
145                 for (size_t vertID = 0; vertID < numV; ++vertID)
146                 {
147                     if (vertID == pID0 || vertID == pID1) continue;
148                     m_UVLList(vertID * 2 + 0) = X(bIndex);
149                     m_UVLList(vertID * 2 + 1) = X(bIndex + numV - numP);
150
151
152

```

```

153     bIndex++;
154 }
155 }
156
157 auto timeEnd = std::chrono::steady_clock::now();
158 auto ms = std::chrono::duration_cast<std::chrono::microseconds>(timeEnd - timeStart)
159 .count() / 1000.0;
160
161 std::cout << "[Free Boundary] LSCM End in " << ms << " ms\n";
162 }
163
164 void FreeBoundarySolver::TutteWithMeanValue(acamcad::polymesh::PolyMesh* mesh)
165 {
166     using acamcad::polymesh::MVert;
167
168     auto timeStart = std::chrono::steady_clock::now();
169
170     auto boundaryVertLists = mesh->boundaryVertices();
171     std::cout << "[Free Boundary] Boundary Points [ " << boundaryVertLists.size() << " ]\n";
172
173     // 1. prepare convex polygon
174     size_t numB = boundaryVertLists.size();
175     size_t numV = mesh->vertices().size();
176
177     // 2. prepare matrix (Eigen::Sparse)
178     Eigen::SparseMatrix<double> A(numV, numV);  A.setZero();
179     Eigen::SparseMatrix<double> x(numV, 2);      x.setZero();
180     Eigen::SparseMatrix<double> b(numV, 2);      b.setZero();
181
182     std::vector<Eigen::Triplet<double>> ATriplets;
183     std::vector<Eigen::Triplet<double>> bTriplets;
184
185     // a) boundary vertices
186     for (auto* pBVert : boundaryVertLists)
187     {
188         size_t vertID = pBVert->index();
189         ATriplets.emplace_back(vertID, vertID, 1.0);
190         bTriplets.emplace_back(vertID, 0, m_UVList(vertID * 2 + 0));
191         bTriplets.emplace_back(vertID, 1, m_UVList(vertID * 2 + 1));
192     }
193
194     // b) inner vertices
195     for (auto* pVert : mesh->vertices())
196     {
197         if (mesh->isBoundary(pVert)) continue;
198
199         size_t vertID = pVert->index();

```

```

200     auto adjVerts = mesh->vertAdjacentVertices(pVert);
201     auto adjWeights = MeanValueWeight(pVert, adjVerts);
202     double weightSUM = std::accumulate(adjWeights.begin(), adjWeights.end(), 0.0);
203
204     for (int j = 0; j < adjVerts.size(); ++j)
205         ATriplets.emplace_back(vertID, adjVerts[j]->index(), adjWeights[j]);
206     ATriplets.emplace_back(vertID, vertID, -weightSUM);
207 }
208
209 A.setFromTriplets(ATriplets.begin(), ATriplets.end());
210 b.setFromTriplets(bTriplets.begin(), bTriplets.end());
211
212 // 3. solve the equation Ax = b
213 Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
214 solver.compute(A);
215 x = solver.solve(b);
216
217 auto timeEnd = std::chrono::steady_clock::now();
218 auto ms = std::chrono::duration_cast<std::chrono::microseconds>(timeEnd - timeStart)
219 ).count() / 1000.0;
220
221 std::cout << "[Free Boundary] Tutte's parameterization end in " << ms << "ms\n";
222
223 for (auto* pVert : mesh->vertices())
224 {
225     if (mesh->isBoundary(pVert)) continue;
226     size_t vertID = pVert->index();
227     m_UVList.segment(vertID * 2, 2) = x.row(vertID);
228 }
229
230 // constrain UV in range [0, 1]x[0, 1]
231 void FreeBoundarySolver::ConstrainUV(acamcad::polymesh::PolyMesh* mesh) const
232 {
233     double xmin = std::numeric_limits<double>::max();
234     double xmax = -std::numeric_limits<double>::max();
235     double ymin = std::numeric_limits<double>::max();
236     double ymax = -std::numeric_limits<double>::max();
237
238     size_t numV = mesh->vertices().size();
239     for (size_t vertID = 0; vertID < numV; ++vertID)
240     {
241         double x = m_UVList(vertID * 2 + 0);
242         double y = m_UVList(vertID * 2 + 1);
243
244         xmin = std::min(xmin, x);
245         xmax = std::max(xmax, x);
246         ymin = std::min(ymin, y);
247         ymax = std::max(ymax, y);

```

```

248 }
249
250     double midx = (xmin + xmax) * 0.5;
251     double midy = (ymin + ymax) * 0.5;
252
253     double maxScale = std::max(xmax - xmin, ymax - ymin);
254
255     for (size_t vertID = 0; vertID < numV; ++vertID)
256     {
257         auto* vert = mesh->vert(vertID);
258         float UVx = static_cast<float>((m_UVList(2 * vertID + 0) - midx) / maxScale);
259         float UVy = static_cast<float>((m_UVList(2 * vertID + 1) - midy) / maxScale);
260         vert->setTexture(UVx * 2.0f, UVy * 2.0f, 0.0);
261     }
262 }
263
264 std::vector<double> FreeBoundarySolver::boundaryEdgesLength(const std::vector<acamcad
265 ::polymesh::MVert*>& boundaries) const
266 {
267     size_t nB = boundaries.size();
268     std::vector<double> lengths(nB, 0.0);
269
270     for (size_t b = 0; b < nB; ++b)
271     {
272         size_t n = (b + 1) % nB;
273         size_t viID = boundaries[b]->index();
274         size_t vjID = boundaries[n]->index();
275
276         Eigen::Vector2d vi;
277         vi << m_UVList(viID * 2), m_UVList(viID * 2 + 1);
278         Eigen::Vector2d vj;
279         vj << m_UVList(vjID * 2), m_UVList(vjID * 2 + 1);
280
281         lengths[b] = (vi - vj).norm();
282     }
283
284     return lengths;
285 }
286
287 std::vector<double> FreeBoundarySolver::MeanValueWeight(acamcad::polymesh::MVert* v,
288 const std::vector<acamcad::polymesh::MVert*>& adjVerts) const
289 {
290     size_t nN = adjVerts.size(); // number of neighbors
291
292     std::vector<double> nWeights(nN, 0.0);
293     std::vector<Eigen::Vector2d> cosAnglesNDistanceSquare(nN, Eigen::Vector2d::Zero());
294
295     size_t v0ID = v->index();

```

```

295     for (size_t i = 0; i < nN; ++i)
296     {
297         size_t j = (i + 1) % nN;
298
299         size_t viID = adjVerts[i]->index();
300         size_t vjID = adjVerts[j]->index();
301
302         Eigen::Vector2d vecA = m_UVList.segment(2 * viID, 2) - m_UVList.segment(2 * vjID,
303         2);
303         Eigen::Vector2d vecB = m_UVList.segment(2 * viID, 2) - m_UVList.segment(2 * v0ID,
304         2);
304         Eigen::Vector2d vecC = m_UVList.segment(2 * vjID, 2) - m_UVList.segment(2 * v0ID,
305         2);
305
306         double a2 = vecA.dot(vecA);
307         double b2 = vecB.dot(vecB);
308         double c2 = vecC.dot(vecC);
309
310         double cosA = (b2 + c2 - a2) / (2.0 * std::sqrt(b2) * std::sqrt(c2));
311         cosAnglesNDistanceSquare[i] << cosA, b2;
312     }
313
314     for (size_t i = 0; i < nN; ++i)
315     {
316         size_t p = (i + nN - 1) % nN;
317
318         double ri = std::sqrt(cosAnglesNDistanceSquare[i].y());
319
320         double cosAi = cosAnglesNDistanceSquare[i].x();
321         double cosAp = cosAnglesNDistanceSquare[p].x();
322         double sinAi = std::sqrt(1.0 - cosAi * cosAi);
323         double sinAp = std::sqrt(1.0 - cosAp * cosAp);
324         double tanAi_2 = sinAi / (1.0 + cosAi);
325         double tanAp_2 = sinAp / (1.0 + cosAp);
326
327         nWeights[i] = (tanAi_2 + tanAp_2) / ri;
328
329         assert(nWeights[i] > 0.0, "negative!");
330     }
331     return nWeights;
332 }
333 }
```

Bibliography

- [1] William Thomas Tutte. “Convex representations of graphs”. In: *Proceedings of the London Mathematical Society* 3.1 (1960), pp. 304–320.
- [2] William Thomas Tutte. “How to draw a graph”. In: *Proceedings of the London Mathematical Society* 3.1 (1963), pp. 743–767.
- [3] Michael S Floater. “Parametrization and smooth approximation of surface triangulations”. In: *Computer aided geometric design* 14.3 (1997), pp. 231–250.
- [4] Bruno Lévy et al. “Least Squares Conformal Maps for Automatic Texture Atlas Generation”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 362–371. ISSN: 0730-0301. doi: [10.1145/566654.566590](https://doi.org/10.1145/566654.566590). URL: <https://doi.org/10.1145/566654.566590>.
- [5] Michael S. Floater. “Mean value coordinates”. In: *Computer Aided Geometric Design* 20.1 (2003), pp. 19–27. ISSN: 0167-8396. doi: [https://doi.org/10.1016/S0167-8396\(03\)00002-5](https://doi.org/10.1016/S0167-8396(03)00002-5). URL: <https://www.sciencedirect.com/science/article/pii/S0167839603000025>.
- [6] Michael S. Floater. “Parametrization and smooth approximation of surface triangulations”. In: *Computer Aided Geometric Design* 14.3 (1997), pp. 231–250. ISSN: 0167-8396. doi: [https://doi.org/10.1016/S0167-8396\(96\)00031-3](https://doi.org/10.1016/S0167-8396(96)00031-3). URL: <https://www.sciencedirect.com/science/article/pii/S0167839696000313>.
- [7] Michael S. Floater. “One-to-One Piecewise Linear Mappings over Triangulations”. In: *Math. Comput.* 72.242 (Apr. 2003), pp. 685–696. ISSN: 0025-5718. doi: [10.1090/S0025-5718-02-01466-7](https://doi.org/10.1090/S0025-5718-02-01466-7). URL: <https://doi.org/10.1090/S0025-5718-02-01466-7>.
- [8] Steven J. Gortler, Craig Gotsman, and Dylan Thurston. “Discrete one-forms on meshes and applications to 3D mesh parameterization”. In: *Computer Aided Geometric Design* 23.2 (2006), pp. 83–112. ISSN: 0167-8396. doi: <https://doi.org/10.1016/j.cagd.2005.05.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0167839605000531>.
- [9] Breannan Smith, Fernando De Goes, and Theodore Kim. “Analytic Eigensystems for Isotropic Distortion Energies”. In: *ACM Trans. Graph.* 38.1 (Feb. 2019). ISSN: 0730-0301. doi: [10.1145/3241041](https://doi.org/10.1145/3241041). URL: <https://doi.org/10.1145/3241041>.