



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 Analytic Eigensystems for Isotropic Distortion Energies	1
1.1 摘要	1
1.2 引言	1
1.3 符号表示	1
1.4 预备知识	2
1.4.1 最优化方法——牛顿法	2
1.4.2 曲面参数化	3
1.4.3 变形梯度 (Deformation Gradient) 与能量函数	4
1.4.4 对记号 vec 的说明	5
1.5 基于能量函数特征系统的海森矩阵半正定投影方法	6
1.5.1 以 \mathbf{S} 为中心的不变量	6
1.5.2 计算对称迪利克雷能量函数对 \mathbf{x} 的偏导数	7
1.5.2.1 局部偏导数到全局偏导数的映射	7
1.5.2.2 计算 $\partial \mathbf{f}_q / \partial \mathbf{x}_q$	8
1.5.2.3 计算 $\partial \Psi_q / \partial \mathbf{f}_q$	9
1.5.2.4 计算 $\partial^2 \Psi_q / \partial \mathbf{f}_q^2$	10
1.6 算法流程	11
1.6.1 投影牛顿法	11
1.6.2 线搜索	12
1.6.3 保证无翻转的最大步长	13
1.7 实验结果	13
1.8 总结	16
附录 A 代码修改说明	17
A.1 对原有文件的修改	17
A.2 新增文件	19
附录 B 致谢	34

第 1 章 Analytic Eigensystems for Isotropic Distortion Energies

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。目前在几何和物理方向存在许多处理非线性形变能量的策略，但是如何使用牛顿类方法计算并达到收敛是一个挑战。本次报告，我们将实现 Smith [1, 2] 等人提出的方法，并使用投影牛顿法计算对称迪利克雷能量下的四组三角形曲面模型的平面参数化结果。

1.2 引言

相较于传统的曲面参数化方法（如 Tutte's Embedding [3–5] 等），基于形变能量的方法可以生成质量更好的参数化结果。但是结果极大地依赖于能量的定义。如果定义的形变能量非凸（函数的海森矩阵不定），那么参数化结果很难使用更快的方法进行优化。Smith [1] 等人使用能量的特征结构，将海森矩阵投影为半正定矩阵，从而适配牛顿法等高阶优化方法。Kim [2] 等人详细地阐述了 Smith 等人的方法的思路、公式及实现细节。本报告主要记录在实现平面参数化的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
\mathbf{x}	向量, $x = [x_1, x_2, \dots, x_n]^\top$
$\bar{\mathbf{x}}$	向量, 指三角形网格模型的顶点坐标
\mathbf{X}	点集, 指网格模型的顶点位置径向集合 $\mathbf{X} = \{\bar{\mathbf{x}}_i = (\bar{x}_i, \bar{y}_i, \bar{z}_i)\}_{i=1}^N$
\mathbf{X}	矩阵, $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m]$
\mathcal{X}	张量, 元素为标量、矩阵或张量
q	Quadrature Point, 插值理论中称为插值节点, 本文称之为基点
$ q $	Volume Weight, 称为体积权重, 语境中与面积、体积等同
\mathbf{F}_q	位于基点 q 处的变形梯度
\mathbf{f}_q	基点 q 处变形梯度 F_q 的向量化 $\mathbf{f}_q = \text{vec}(\mathbf{F}_q)$
$\text{vec}(\mathbf{A})$	矩阵的向量化, 详见 1.4.4 节
$\text{vec}(\mathcal{A})$	张量的向量化, 详见 1.4.4 节
$\text{tr}(\mathbf{A})$	矩阵的迹
$\det(\mathbf{A})$	矩阵的行列式, 等同于 $ A $
$\Psi(\mathbf{x})$	定义在 \mathbf{x} 上的能量函数
$\Psi_q(\mathbf{F}_q)$	定义在基点 q 的形变梯度 \mathbf{F}_q 上的能量函数
σ_i	变形梯度 \mathbf{F}_q 的主拉伸量 $\{\sigma_i\}$
I_i	S-中心不变量
λ_i	特征系统中的特征值
\mathbf{e}_i	特征系统中的特征向量

1.4 预备知识

我们部分采用文章 [1] 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 ($U_b = \{P_1, P_2, \dots, P_n\}$)。

注 注意 $\bar{\mathbf{x}}$ 表示输入三角网格的顶点位置，而不是每一次迭代过程中，来自上一次迭代结果的顶点位置。在优化的过程中 $\bar{\mathbf{x}}$ 保持不变。

1.4.1 最优化方法——牛顿法

本节内容主要来自于 [6]。

定义 1.1 (牛顿方程与牛顿方向)

我们考虑最优化问题如下：

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \quad \text{s.t. } \mathbf{x} \in \Omega, \quad (1.1)$$

其中 $\Omega \in \mathbb{R}^n$ 是可行域。如果 $f(\mathbf{x})$ 是二次连续可微的，那么 $f(\mathbf{x})$ 在迭代点 \mathbf{x}^k 处的二阶泰勒展开式是：

$$f(\mathbf{x}^k + \mathbf{d}^k) = f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)^\top \mathbf{d}^k + \frac{1}{2} (\mathbf{d}^k)^\top \nabla^2 f(\mathbf{x}^k) \mathbf{d}^k + o(\|\mathbf{d}^k\|). \quad (1.2)$$

为求取合适的下降方向 \mathbf{d}^k ，我们忽略上式中的高阶项，并求忽略高阶项后公式 1.2 关于 \mathbf{d}^k 的稳定点^a。稳定点 \mathbf{d}^k 满足：

$$\nabla^2 f(\mathbf{x}^k) \mathbf{d}^k = -\nabla f(\mathbf{x}^k). \quad (1.3)$$

上式也被称为牛顿方程^b。当 $\nabla^2 f(\mathbf{x}^k)$ 非奇异时，可得

$$\mathbf{d}^k = -[\nabla^2 f(\mathbf{x}^k)]^{-1} \nabla f(\mathbf{x}^k), \quad (1.4)$$

一般称满足上式的 \mathbf{d}^k 为牛顿方向。

^a我们将满足 $\nabla f(\mathbf{x}) = 0$ 的点称为稳定点，也称为驻点或者临界点

^b如果 $f(\mathbf{x})$ 是个向量值函数，那么 $\nabla^2 f(\mathbf{x})$ 是 $n \times n$ 的矩阵。



根据 1.4 我们可以得到经典牛顿法¹的迭代点更新格式为

$$\mathbf{x}^{k+1} = \mathbf{x}^k - [\nabla^2 f(\mathbf{x}^k)]^{-1} \nabla f(\mathbf{x}^k). \quad (1.5)$$

经典牛顿法有很好的局部收敛性质：

1. 如果初始点离问题的解 \mathbf{x}^* 足够近，则牛顿法产生的迭代点到 $\{\mathbf{x}^k\}$ 收敛到 \mathbf{x}^* ；
2. $\{\mathbf{x}^k\}$ 收敛到 \mathbf{x}^* 的速度是 Q —二次的；
3. $\{\|\nabla f(\mathbf{x}^k)\|\}$ Q —二次收敛到 0。

但是经典牛顿法也存在一些弊端：

1. 初始迭代点 \mathbf{x}^0 必须距离问题的解 \mathbf{x}^* 充分近，即局部收敛性；
2. 要求海森矩阵 $\nabla^2 f(\mathbf{x}^*)$ 正定。如果 $\nabla^2 f(\mathbf{x}^*)$ 是奇异的半正定矩阵，收敛速度可能不足；
3. 对于病态问题，牛顿法的收敛域 Ω 可能会变小。

所以在实际应用中，人们往往使用牛顿法对初步的、低精度的解进行优化，从而得到高精度的解。

¹牛顿法中的迭代点步长为 α_k 。如果 $\alpha_k = 1$ 我们称之为经典牛顿法，即公式 1.5 表示的形式。

1.4.2 曲面参数化

根据前一次的作业，我们知道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 $S = (G, X)$, $G = G(V, E, F)$, $X = \{\bar{x}_i = (\bar{x}_i, \bar{y}_i, \bar{z}_i)\}_{i=1}^{N_V}$, 平面参数化方法是找到一个映射:

$$f : \bar{x}_i = (\bar{x}_i, \bar{y}_i, \bar{z}_i)^T \mapsto P_i = (u_i, v_i).$$

我们希望参数化结果 $\mathcal{P} = \mathcal{P}(G, U_b)$ 满足良好的性质，包括无翻转（Flip-free），局部单射（Locally injective）和全局单射（Globally-injective）。此外，我们还希望映射 f 是等距映射、共性映射或者保面积映射。我们使用 Tutte's Embedding [3–5] 的方法找到了满足无翻转的参数化结果（图 1.1），此结果可以作为牛顿法的输入，从而得到精度更高、质量更好的参数化结果。

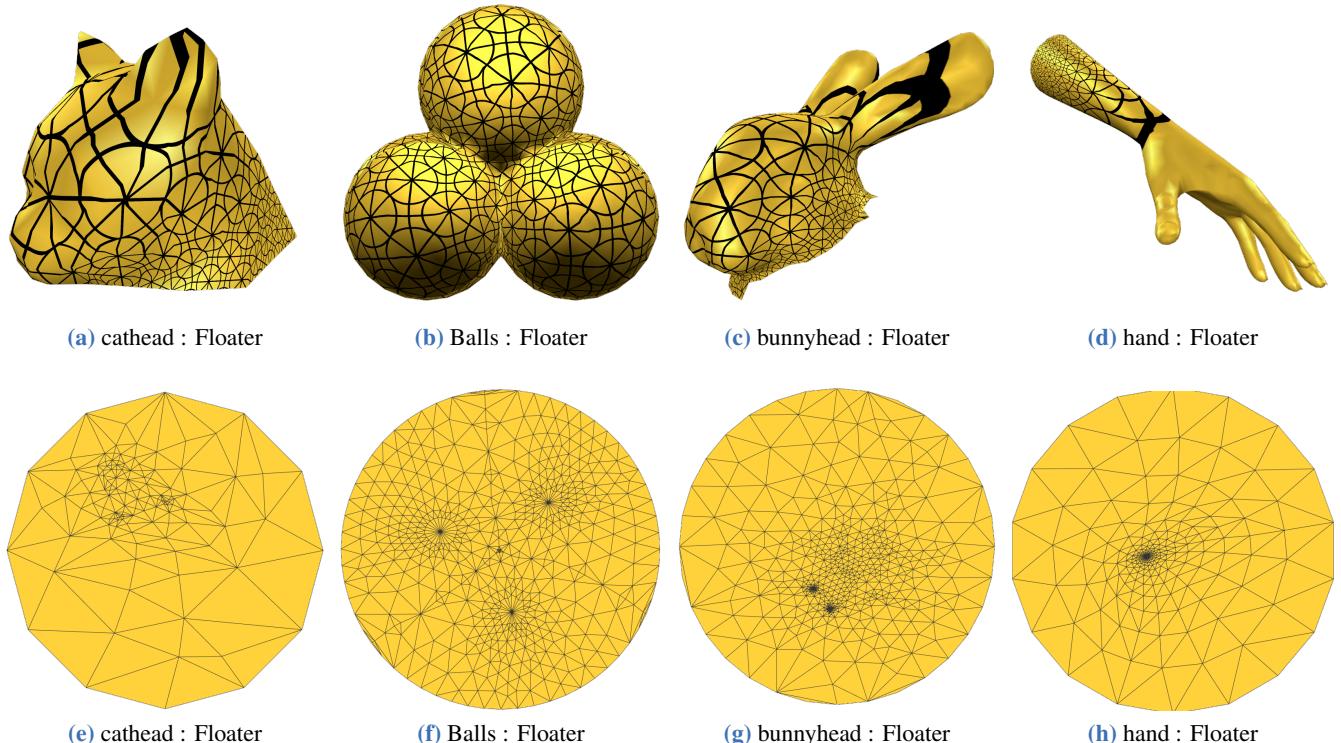


图 1.1: Tutte's Embedding: Floater's weight 结果（第一排为纹理映射贴图绘制结果，第二排为与第一排对应的平面参数化结果。边界形状：正多边形）

从图 1.1 中可以看出，Tutte 的方法存在以下几个问题：

1. 三角形变形严重。Tutte's Weight [3, 4] 没有考虑三角形的形状，而 Floater's weight [5] 虽然将三角形的形状考虑在内，但无法打破 Tutte's Embedding 框架的局限。
2. 距离边缘较远的顶点呈现密集堆积的趋势（如图 1.1(c, g) 中 bunnyhead 数据的耳朵部分，和图 1.1(d, h) hand 数据的手指部分）。密集堆积的顶点容易导致三角形在计算精度不足的情况下退化，且密集区域会导致纹理映射出现极大的变形与扭曲。

为了处理这种问题，人们提出了许多从优化能量函数的角度优化参数化网格质量的方法 [1]，他们都在解决针对各种能量函数的最优化问题。一阶方法、二阶方法等线搜索类方法 [6] 是优化能量函数的常用方法。但是此方法对能量函数提出了严苛的条件，学者们提出了许多的能量函数和改进方案，以适配合适的线搜索类方法。本文要实现的文章 [1] 分析了各向同性形变能量函数的特征系统，解析地将海森矩阵投影为半正定矩阵，从而适配牛顿法等最优化方法。

1.4.3 变形梯度 (Deformation Gradient) 与能量函数

变形梯度的概念来自于连续介质力学 [7]。以二维空间中的可变形三角形 $\bar{T} = \{\bar{v}_1, \bar{v}_2, \bar{v}_3\}$ 为例, \bar{v}_i 是 \bar{T} 的顶点, \bar{x}_i 是 v_i 的坐标。变形后的三角形为 $T = \{v_1, v_2, v_3\}$, v_i 是 T 的顶点, x_i 是 v_i 的坐标。对于 \bar{T} 和 T 内的微元 $d\bar{x}$ 和 dx 满足 $dx = (\partial \bar{x} / \partial x)d\bar{x} = F(\bar{x})d\bar{x}$, F 称为变形梯度。假设 \bar{T} 的逐顶点的变形映射 $\phi_{\bar{T}}$ 是分片线性的, 那么 $\phi_{\bar{T}}$ 对于 x_i 有:

$$x_i = \phi_{\bar{T}}(\bar{x}_i) = F\bar{x}_i + b, \quad i = 1, 2, 3. \quad (1.6)$$

并且:

$$\frac{\partial x_i}{\partial \bar{x}_i} = \frac{\partial \phi_{\bar{T}}(\bar{x}_i)}{\partial \bar{x}_i} = \frac{\partial}{\partial \bar{x}_i}(F\bar{x}_i + b) = F.$$

如图 1.2, 在变形映射 ϕ 下, 三角形 \bar{T} 变形为 T 。对于 $\phi_{\bar{T}}$, 根据公式 1.6, 显然有如下关系成立:

$$\begin{cases} x_1 = \phi_{\bar{T}}(\bar{x}_1) = F\bar{x}_1 + b, \\ x_2 = \phi_{\bar{T}}(\bar{x}_2) = F\bar{x}_2 + b, \\ x_3 = \phi_{\bar{T}}(\bar{x}_3) = F\bar{x}_3 + b, \end{cases} \Rightarrow F \begin{bmatrix} \bar{x}_2 - \bar{x}_1 & \bar{x}_3 - \bar{x}_1 \\ \hline D_m = [d_{m_1}, d_{m_2}] \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ \hline D_s = [d_{s_1}, d_{s_2}] \end{bmatrix}$$

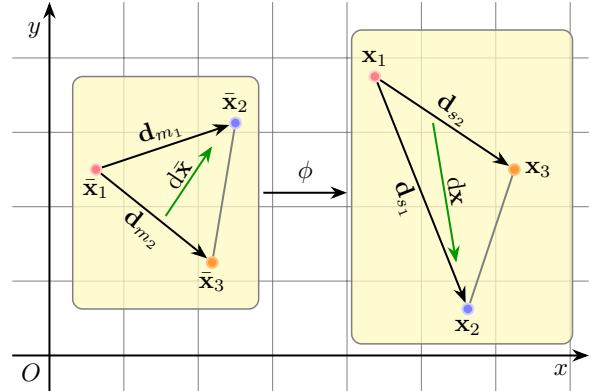


图 1.2: 变形前后的三角形

根据上式可得

$$F = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ \hline D_s = [d_{s_1}, d_{s_2}] \end{bmatrix} \begin{bmatrix} \bar{x}_2 - \bar{x}_1 & \bar{x}_3 - \bar{x}_1 \\ \hline D_m = [d_{m_1}, d_{m_2}] \end{bmatrix}^{-1} = D_s D_m^{-1}. \quad (1.7)$$

对于离散化的三角形网格曲面 $\mathcal{P} = \mathcal{P}(G, U_b)$ 每一个三角面 T 的三个顶点位置可以通过定义在基点 q 上的基函数插值得到。一般我们将基点 q 定义在三角形、四边形等单纯形的几何中心处。此时, 我们可以使用基点的符号 q 代替符号 T , 从而将形变梯度 F 改写为 F_q 。 F_q 取决于定义在 q 处的基函数的选取方式。公式 1.7 可改写为基点表示的形式 $F_q = D_s D_m^{-1}$ 。

注 使用基点 q 的表示与基于顶点的表示在计算上没有区别。除了计算 $|q|$ 外, q 只作为下标, 起到“区分不同的三角面”的作用。 q 的选取不影响结果, 计算时仍使用三角形的顶点位置和边向量。

基于 F 我们可以定义许多各向异性的能量函数, 如

能量	全称	能量形式
ARAP [8]	As-Rigid-As-Possible	$\Psi_{\text{ARAP}} = \ F - R\ $
SD [9]	Symmetric Dirichlet	$\Psi_{\text{SD}} = \ F\ ^2 + \ F^{-1}\ ^2$
MIPS [10]	Most Isometric ParameterizationS	$\Psi_{\text{MIPS}} = \text{tr}(F^\top F) / \det(F)$

表 1.2: 定义在 F 上的各向异性能量

大部分能量的海森矩阵是不定的。Smith [1] 等人提出的方法可以将这些各向异性能量的海森矩阵投影为半正定矩阵, 从而与牛顿法适配。三角形网格的能量 Ψ 定义在顶点位置 x 上, 也是所有三角面的能量之和, 即

$$\Psi(x) = \sum_q \Psi_q(F_q) |q|. \quad (1.8)$$

1.4.4 对记号 vec 的说明

对于 2 阶和 4 阶的张量, Smith [1] 等人引入了 vec 操作。它可以完成对矩阵和张量的“向量化”, 对于矩阵 \mathbf{A} , 有

$$\mathbf{A} = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \iff \text{vec}(\mathbf{A}) = [a, b, c, d]^\top = \mathbf{a}.$$

对于张量 \mathbf{A}

$$\mathcal{A} = \begin{bmatrix} \begin{bmatrix} a & c \\ b & d \end{bmatrix} & \begin{bmatrix} i & k \\ j & l \end{bmatrix} \\ \begin{bmatrix} e & g \\ f & h \end{bmatrix} & \begin{bmatrix} m & o \\ n & p \end{bmatrix} \end{bmatrix} = \begin{bmatrix} [\mathbf{A}_{11}] & [\mathbf{A}_{12}] \\ [\mathbf{A}_{21}] & [\mathbf{A}_{22}] \end{bmatrix},$$

有

$$\text{vec}(\mathcal{A}) = [\text{vec}(\mathbf{A}_{11}) | \text{vec}(\mathbf{A}_{21}) | \text{vec}(\mathbf{A}_{12}) | \text{vec}(\mathbf{A}_{22})] = \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix} = \mathbf{A}.$$

除此之外, 令 $\mathbf{a} = \text{vec}(\mathbf{A})$, 则有

$$\text{vec}\left(\frac{\partial \mathbf{A}}{\partial x}\right) = \text{vec}\left(\begin{bmatrix} \frac{\partial a_{11}}{\partial x} & \cdots & \frac{\partial a_{1n}}{\partial x} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_{m1}}{\partial x} & \cdots & \frac{\partial a_{mn}}{\partial x} \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial a_{11}}{\partial x} \\ \vdots \\ \frac{\partial a_{mn}}{\partial x} \end{bmatrix} = \frac{\partial \text{vec}(\mathbf{A})}{\partial x}$$

$$\text{vec}\left(\frac{\partial a}{\partial \mathbf{X}}\right) = \text{vec}\left(\begin{bmatrix} \frac{\partial a}{\partial x_{11}} & \cdots & \frac{\partial a}{\partial x_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial a}{\partial x_{m1}} & \cdots & \frac{\partial a}{\partial x_{mn}} \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial a}{\partial x_{11}} \\ \vdots \\ \frac{\partial a}{\partial x_{mn}} \end{bmatrix} = \frac{\partial a}{\partial \text{vec}(\mathbf{X})}$$

$$\begin{aligned} \text{vec}\left(\frac{\partial \mathbf{A}}{\partial \mathbf{x}}\right) &= \text{vec}\left(\begin{bmatrix} \begin{bmatrix} \frac{\partial a_{11}}{\partial \mathbf{x}} \end{bmatrix} & \cdots & \begin{bmatrix} \frac{\partial a_{1n}}{\partial \mathbf{x}} \end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} \frac{\partial a_{m1}}{\partial \mathbf{x}} \end{bmatrix} & \cdots & \begin{bmatrix} \frac{\partial a_{mn}}{\partial \mathbf{x}} \end{bmatrix} \end{bmatrix}\right) \\ &= \left[\text{vec}\left(\frac{\partial a_{11}}{\partial \mathbf{x}}\right) \middle| \cdots \middle| \text{vec}\left(\frac{\partial a_{m1}}{\partial \mathbf{x}}\right) \middle| \cdots \middle| \text{vec}\left(\frac{\partial a_{1n}}{\partial \mathbf{x}}\right) \middle| \cdots \middle| \text{vec}\left(\frac{\partial a_{mn}}{\partial \mathbf{x}}\right) \right] \\ &= \begin{bmatrix} \frac{\partial a_{11}}{\partial x_1} & \cdots & \frac{\partial a_{mn}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_{11}}{\partial x_p} & \cdots & \frac{\partial a_{mn}}{\partial x_p} \end{bmatrix} = \frac{\partial \text{vec}(\mathbf{A})^\top}{\partial \mathbf{x}} \end{aligned}$$

$$\text{vec}\left(\frac{\partial \mathbf{a}}{\partial \mathbf{X}}\right) = \text{vec}\left(\begin{bmatrix} \begin{bmatrix} \frac{\partial a_1}{\partial \mathbf{X}} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \frac{\partial a_p}{\partial \mathbf{X}} \end{bmatrix} \end{bmatrix}\right) = \left[\text{vec}\left(\frac{\partial a_1}{\partial \mathbf{X}}\right) \middle| \cdots \middle| \text{vec}\left(\frac{\partial a_p}{\partial \mathbf{X}}\right) \right] = \begin{bmatrix} \frac{\partial a_1}{\partial x_{11}} & \cdots & \frac{\partial a_p}{\partial x_{11}} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial x_{mn}} & \cdots & \frac{\partial a_p}{\partial x_{mn}} \end{bmatrix} = \frac{\partial \mathbf{a}^\top}{\partial \text{vec}(\mathbf{X})}$$

需要注意的是上述关于标量、矩阵和张量的形式会随着分子布局和分母布局的不同而变化。

1.5 基于能量函数特征系统的海森矩阵半正定投影方法

我们以三维空间的三角型网格模型 $S = S(G, \mathbf{X})$, 为例, 最终的目标是得到平面的参数化结果 $\mathcal{P} = \mathcal{P}(G, U_b)$, 顶点数量 $N_V = |\mathbf{V}|$, 本征维度 $d = 2$, 自由度 $n = dN_V$ 。在介绍能量函数的特征系统之前, 我们先考虑对形变梯度 \mathbf{F} 进行一些变换。

根据 SVD 方法, 变形梯度 \mathbf{F} 可以分解为 $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^\top$, 其中 \mathbf{U} , \mathbf{V} 是二维的特殊正交群 $\text{SO}(d)$ 中的正交矩阵, Σ 是包含主拉伸量 σ_1, σ_2 的对角矩阵。使用 \mathbf{F} 的 SVD 结果, 我们可以构造 \mathbf{F} 的极分解 $\mathbf{F} = \mathbf{RS}$, 其中 $\mathbf{R} = \mathbf{UV}^\top$ 是旋转矩阵, $\mathbf{S} = \mathbf{V}\Sigma\mathbf{V}^\top$ 。

注 $\Sigma = \text{diag}(\sigma_1, \sigma_2)$ 中的主拉伸量可能出现负值, 表示三角形在形变过程中发生了翻转。实际计算中, 需要根据拉伸量的符号对 SVD 分解的结果 $\{\mathbf{U}, \Sigma, \mathbf{V}\}$ 进行翻转校正。这种方法也叫做带符号的 SVD 分解 (SSVD) 方法。

我们要求对变形梯度 \mathbf{F} 进行极分解 $\mathbf{F} = \mathbf{RS}$ 时, 矩阵 \mathbf{R} 是纯旋转矩阵。如果三角形出现翻转, 则 SVD 方法分解 \mathbf{F} 得到的矩阵 Σ 中的最小特征值会变为 -1 。这种情况下 \mathbf{F} 的极分解结果 $\{\mathbf{R}, \mathbf{S}\}$ 中, 矩阵 R 不是纯旋转矩阵。为校正翻转的情况, 我们构建反射矩阵 \mathbf{L} 如下:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ 0 & \det(\mathbf{UV}^\top) \end{bmatrix}$$

并对 $\{\mathbf{U}, \Sigma, \mathbf{V}\}$ 进行变换:

$$\begin{aligned} \Sigma &\leftarrow \Sigma\mathbf{L}, \\ \mathbf{U} &\leftarrow \begin{cases} \mathbf{UL}, & \text{如果 } \det\mathbf{U} < 0 \text{ 并且 } \det\mathbf{V} > 0 \\ \mathbf{U}, & \text{其它} \end{cases} \\ \mathbf{V} &\leftarrow \begin{cases} \mathbf{VL}, & \text{如果 } \det\mathbf{U} > 0 \text{ 并且 } \det\mathbf{V} < 0 \\ \mathbf{U}, & \text{其它} \end{cases} \end{aligned}$$

根据变换后的 $\{\mathbf{U}, \Sigma, \mathbf{V}\}$ 可以得到极分解 $\mathbf{R} = \mathbf{UV}^\top$, $\mathbf{S} = \mathbf{V}\Sigma\mathbf{V}^\top$, 其中 \mathbf{R} 是纯旋转矩阵。

1.5.1 以 \mathbf{S} 为中心的不变量

大多数几何优化中的变形能量 Ψ 是各向同性的能量, 可以使用从 \mathbf{F} 导出的不变量表示。

定义 1.2 (以 \mathbf{S} 为中心的不变量)

考虑 \mathbf{F} 的极分解 $\mathbf{F} = \mathbf{RS}$, 有不变量如下:

$$\left\{ \begin{array}{l} I_1 = \text{tr}(\mathbf{S}) = \sum_i \sigma_i, \end{array} \right. \quad (1.9a)$$

$$\left\{ \begin{array}{l} I_2 = \|\mathbf{S}\|^2 = \sum_i \sigma_i^2, \end{array} \right. \quad (1.9b)$$

$$\left\{ \begin{array}{l} I_3 = \det(\mathbf{S}) = \prod_i \sigma_i. \end{array} \right. \quad (1.9c)$$

其中 I_1 用于表示能量中的线性系统, I_2 用于衡量最小二乘扭曲, I_3 用于衡量 \mathbf{F} 对体积的改变。

不变量在描述 \mathbf{F} 和能量等有明显优势, 用于分析各向异性能量的特征系统, 并且在每个基点 q 对应的三角形内, 把基于 \mathbf{F}_q 的能量函数变为使用 $\{I_1, I_2, I_3\}$ 表示的能量函数。在 2D 情况下有 $I_2 = I_1^2 - 2I_3$ 。

1.5.2 计算对称迪利克雷能量函数对 \mathbf{x} 的偏导数

我们使用链式法则，并考虑中间变量 $\mathbf{f}_q = \text{vec}(\mathbf{F}_q)$ ，计算 1.8 对于 \mathbf{x} 的一阶和二阶偏导数如下：

$$\frac{\partial \Psi}{\partial \mathbf{x}} = \sum_q |q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \frac{\partial \Psi_q}{\partial \mathbf{f}_q}, \quad (1.10a)$$

$$\frac{\partial^2 \Psi}{\partial \mathbf{x}^2} = \sum_q |q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}}. \quad (1.10b)$$

$|q|$ 是体积权重，一般指三角形（四面体）的有向面积（体积）， $|q| = 1/2 \det(\mathbf{D}_m)$ ($|q| = 1/6 \det(\mathbf{D}_m)$)。

注 在公式 1.10a 和 1.10b 中， \mathbf{x} 代表任意顶点的坐标。但是等号右端包含了所有基点 q 上的能量函数 $\Psi_q(\mathbf{F}_q)$ 对 \mathbf{x} 的导数之和，而不仅仅是包含点 \mathbf{x} 的三角形上能量函数对 \mathbf{x} 的导数。这是因为每个三角形基点 q 上选取的、用于对顶点位置插值的基函数的定义域遍布整个三角形曲面 S ，顶点 \mathbf{x} 会对所有定义在三角面的基点 q 上的能量函数 $\Psi_q(\mathbf{F}_q)$ 有贡献。

此外，在具体计算的过程中，我们会一次性计算出能量函数 $\Psi(\mathbf{x})$ 对所有顶点 $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N_V}$ 的导数和二阶导数。此时有 \mathbf{x} 大小为 $n \times 1$ ：

$$\mathbf{x} = [\mathbf{x}_{1.x}, \mathbf{x}_{1.y}, \dots, \mathbf{x}_{N_V.x}, \mathbf{x}_{N_V.y}]^\top, \quad (1.11)$$

所以公式 1.10a 中， $\partial \Psi / \partial \mathbf{x}$ 大小为 $n \times 1$ ， $\partial \mathbf{f}_q / \partial \mathbf{x}$ 大小为 $d^2 \times n$ ， $\partial \Psi_q / \partial \mathbf{f}_q$ 大小为 $d^2 \times 1$ 。公式 1.10b 中， $\partial^2 \Psi / \partial \mathbf{x}^2$ 大小为 $n \times n$ ， $\partial^2 \Psi_q / \partial \mathbf{f}_q^2$ 大小为 $d^2 \times d^2$ 。

1.5.2.1 局部偏导数到全局偏导数的映射

通常在计算 1.10a 和 1.10b 的过程中，我们可以预先计算出对于每个三角形上的能量函数对三角形顶点 $\mathbf{x}_q = [\mathbf{x}_{q1.x}, \mathbf{x}_{q1.y}, \mathbf{x}_{q2.x}, \mathbf{x}_{q2.y}, \mathbf{x}_{q3.x}, \mathbf{x}_{q3.y}]^\top$ 的一阶和二阶偏导数

$$\left\{ \begin{array}{l} \frac{\partial \Psi_q}{\partial \mathbf{x}_q} = |q| \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}_q}^\top \frac{\partial \Psi_q}{\partial \mathbf{f}_q}, \\ \frac{\partial^2 \Psi}{\partial \mathbf{x}_q^2} = |q| \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}_q}^\top \frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}_q}. \end{array} \right. \quad (1.12a)$$

$$\left\{ \begin{array}{l} \frac{\partial \Psi_q}{\partial \mathbf{x}_q} = |q| \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}_q}^\top \frac{\partial \Psi_q}{\partial \mathbf{f}_q}, \\ \frac{\partial^2 \Psi}{\partial \mathbf{x}_q^2} = |q| \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}_q}^\top \frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}_q}. \end{array} \right. \quad (1.12b)$$

其中 q_1, q_2 和 q_3 是基点 q 所在三角形的三个顶点的指标，展开有：

$$\left[\begin{array}{c} q \frac{\partial \Psi_q}{\partial \mathbf{x}_q} \\ q_1.x \frac{\partial \Psi_q}{\partial \mathbf{x}_{q1.x}} \\ q_1.y \frac{\partial \Psi_q}{\partial \mathbf{x}_{q1.y}} \\ q_2.x \frac{\partial \Psi_q}{\partial \mathbf{x}_{q2.x}} \\ q_2.y \frac{\partial \Psi_q}{\partial \mathbf{x}_{q2.y}} \\ q_3.x \frac{\partial \Psi_q}{\partial \mathbf{x}_{q3.x}} \\ q_3.y \frac{\partial \Psi_q}{\partial \mathbf{x}_{q3.y}} \end{array} \right] = \left[\begin{array}{cccccc} q_1.x & q_1.y & q_2.x & q_2.y & q_3.x & q_3.y \\ \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.x}^2} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.x} \partial \mathbf{x}_{q1.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.x} \partial \mathbf{x}_{q2.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.x} \partial \mathbf{x}_{q2.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.x} \partial \mathbf{x}_{q3.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.x} \partial \mathbf{x}_{q3.y}} \\ \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.y} \partial \mathbf{x}_{q1.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.y}^2} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.y} \partial \mathbf{x}_{q2.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.y} \partial \mathbf{x}_{q2.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.y} \partial \mathbf{x}_{q3.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q1.y} \partial \mathbf{x}_{q3.y}} \\ \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.x} \partial \mathbf{x}_{q1.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.x} \partial \mathbf{x}_{q1.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.x}^2} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.x} \partial \mathbf{x}_{q2.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.x} \partial \mathbf{x}_{q3.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.x} \partial \mathbf{x}_{q3.y}} \\ \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.y} \partial \mathbf{x}_{q1.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.y} \partial \mathbf{x}_{q1.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.y} \partial \mathbf{x}_{q2.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.y}^2} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.y} \partial \mathbf{x}_{q3.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q2.y} \partial \mathbf{x}_{q3.y}} \\ \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.x} \partial \mathbf{x}_{q1.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.x} \partial \mathbf{x}_{q1.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.x} \partial \mathbf{x}_{q2.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.x}^2} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.x} \partial \mathbf{x}_{q2.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.x} \partial \mathbf{x}_{q3.y}} \\ \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.y} \partial \mathbf{x}_{q1.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.y} \partial \mathbf{x}_{q1.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.y} \partial \mathbf{x}_{q2.x}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.y}^2} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.y} \partial \mathbf{x}_{q2.y}} & \frac{\partial^2 \Psi_q}{\partial \mathbf{x}_{q3.y}^2} \end{array} \right] \begin{array}{c} q \\ q_1.x \\ q_1.y \\ q_2.x \\ q_2.y \\ q_3.x \\ q_3.y \end{array}$$

根据指标 $[q_1.x, q_1.y, q_2.x, q_2.y, q_3.x, q_3.y]^\top$ 与公式 1.11 中 \mathbf{x} 的对应关系，将 $\partial \Psi_q / \partial \mathbf{x}_q$ 和 $\partial^2 \Psi_q / \partial \mathbf{x}_q^2$

的元素加到 $\partial\Psi/\partial\mathbf{x}$ 和 $\partial^2\Psi/\partial\mathbf{x}^2$ 的对应位置:

$$\begin{array}{c} q_1.x \\ \downarrow \\ \frac{\partial\Psi_q}{\partial\mathbf{x}_{q_1x}} \end{array} \quad \begin{array}{c} q_1.y \\ \downarrow \\ \frac{\partial\Psi_q}{\partial\mathbf{x}_{q_1y}} \end{array} \quad \begin{array}{c} q_2.x \\ \downarrow \\ \frac{\partial\Psi_q}{\partial\mathbf{x}_{q_2x}} \end{array} \quad \begin{array}{c} q_2.y \\ \downarrow \\ \frac{\partial\Psi_q}{\partial\mathbf{x}_{q_2y}} \end{array} \quad \begin{array}{c} q_3.x \\ \downarrow \\ \frac{\partial\Psi_q}{\partial\mathbf{x}_{q_3x}} \end{array} \quad \begin{array}{c} q_3.y \\ \downarrow \\ \frac{\partial\Psi_q}{\partial\mathbf{x}_{q_3y}} \end{array} \quad \dots$$

$$\left[\dots \begin{array}{cccccc} q_1.x & q_1.y & q_2.x & q_2.y & q_3.x & q_3.y \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1x}^2} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1x}\partial\mathbf{x}_{q_1y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1x}\partial\mathbf{x}_{q_2x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1x}\partial\mathbf{x}_{q_2y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1x}\partial\mathbf{x}_{q_3x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1x}\partial\mathbf{x}_{q_3y}} & \dots \\ \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1y}\partial\mathbf{x}_{q_1x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1y}^2} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1y}\partial\mathbf{x}_{q_2x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1y}\partial\mathbf{x}_{q_2y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1y}\partial\mathbf{x}_{q_3x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_1y}\partial\mathbf{x}_{q_3y}} & \dots \\ \dots & \dots \\ \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2x}\partial\mathbf{x}_{q_1x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2x}\partial\mathbf{x}_{q_1y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2x}^2} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2x}\partial\mathbf{x}_{q_2y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2x}\partial\mathbf{x}_{q_3x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2x}\partial\mathbf{x}_{q_3y}} & \dots \\ \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2y}\partial\mathbf{x}_{q_1x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2y}\partial\mathbf{x}_{q_1y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2y}\partial\mathbf{x}_{q_2x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2y}^2} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2y}\partial\mathbf{x}_{q_3x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_2y}\partial\mathbf{x}_{q_3y}} & \dots \\ \dots & \dots \\ \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3x}\partial\mathbf{x}_{q_1x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3x}\partial\mathbf{x}_{q_1y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3x}\partial\mathbf{x}_{q_2x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3x}\partial\mathbf{x}_{q_2y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3x}^2} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3x}\partial\mathbf{x}_{q_3y}} & \dots \\ \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3y}\partial\mathbf{x}_{q_1x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3y}\partial\mathbf{x}_{q_1y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3y}\partial\mathbf{x}_{q_2x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3y}\partial\mathbf{x}_{q_2y}} & \dots & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3y}\partial\mathbf{x}_{q_3x}} & \frac{\partial^2\Psi_q}{\partial\mathbf{x}_{q_3y}^2} & \dots \\ \dots & \dots \end{array} \right]^\top$$

$\leftarrow q_1.x$
 $\leftarrow q_1.y$
 $\leftarrow q_2.x$
 $\leftarrow q_2.y$
 $\leftarrow q_3.x$
 $\leftarrow q_3.y$

除此之外，也可以计算 $\partial\mathbf{f}_q/\partial\mathbf{x}$ 而不是 $\partial\mathbf{f}_q/\partial\mathbf{x}_q$ ，并使用 $\partial\mathbf{f}_q/\partial\mathbf{x}$ 参与计算，效果与上述方法等价。

1.5.2.2 计算 $\partial\mathbf{f}_q/\partial\mathbf{x}_q$

根据章节 1.4.4 中对 vec 的介绍，可知

$$\text{vec} \left(\frac{\partial\mathbf{F}_q}{\partial\mathbf{x}_q} \right) = \frac{\partial \text{vec}(\mathbf{F}_q)^\top}{\partial\mathbf{x}_q} = \frac{\partial\mathbf{f}_q^\top}{\partial\mathbf{x}_q},$$

将公式 1.7 两边对 \mathbf{x}_q 求导，有

$$\frac{\partial\mathbf{F}}{\partial(\mathbf{D}_s\mathbf{D}_m^{-1})} = \frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_q}\mathbf{D}_m^{-1}. \quad (1.13)$$

参考 [2] 中附录 E 推导 $\partial\mathbf{F}/\partial\mathbf{x}$ 的方法，类似地我们可以推出：

$$\frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_q} = \left[\left[\frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_1.x}} \right], \left[\frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_1.y}} \right], \left[\frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_2.x}} \right], \left[\frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_2.y}} \right], \left[\frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_3.x}} \right], \left[\frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_3.y}} \right] \right]^\top, \quad (1.14)$$

并且

$$\begin{aligned} \frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_1.x}} &= \begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix}, & \frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_2.x}} &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & \frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_3.x}} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \\ \frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_1.y}} &= \begin{bmatrix} 0 & 0 \\ -1 & -1 \end{bmatrix}, & \frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_2.y}} &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, & \frac{\partial\mathbf{D}_s}{\partial\mathbf{x}_{q_3.y}} &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}. \end{aligned}$$

此外，把矩阵 \mathbf{D}_m^{-1} 按照列向量和行向量的形式表示如下：

$$\mathbf{D}_m^{-1} = \begin{bmatrix} -\mathbf{r}_1 - \\ -\mathbf{r}_2 - \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}.$$

公式 1.13 的计算规则为 $\partial \mathbf{F}_q / \partial \mathbf{x}_i = (\partial \mathbf{D}_s / \partial \mathbf{x}_i) \mathbf{D}_m^{-1}$ ，于是有

$$\begin{aligned} \frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_1}.x} &= \begin{bmatrix} -(c_{11} + c_{21}) & -(c_{12} + c_{22}) \\ 0 & 0 \end{bmatrix}, & \frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_2}.x} &= \begin{bmatrix} -\mathbf{r}_1 - \\ -\mathbf{0} - \end{bmatrix}, & \frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_3}.x} &= \begin{bmatrix} -\mathbf{0} - \\ -\mathbf{r}_1 - \end{bmatrix}, \\ \frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_1}.y} &= \begin{bmatrix} 0 & 0 \\ -(c_{11} + c_{21}) & -(c_{12} + c_{22}) \end{bmatrix}, & \frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_2}.y} &= \begin{bmatrix} -\mathbf{r}_2 - \\ -\mathbf{0} - \end{bmatrix}, & \frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_3}.y} &= \begin{bmatrix} -\mathbf{0} - \\ -\mathbf{r}_2 - \end{bmatrix}. \end{aligned}$$

则

$$\begin{aligned} \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}_q} &= \text{vec} \left(\left[\left[\frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_1}.x} \right], \left[\frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_1}.y} \right], \left[\frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_2}.x} \right], \left[\frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_2}.y} \right], \left[\frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_3}.x} \right], \left[\frac{\partial \mathbf{F}_q}{\partial \mathbf{x}_{q_3}.y} \right] \right]^\top \right) \\ &= \left[\frac{\partial \text{vec}(\mathbf{F}_q)}{\partial \mathbf{x}_{q_1}.x} \middle| \frac{\partial \text{vec}(\mathbf{F}_q)}{\partial \mathbf{x}_{q_1}.y} \middle| \frac{\partial \text{vec}(\mathbf{F}_q)}{\partial \mathbf{x}_{q_2}.x} \middle| \frac{\partial \text{vec}(\mathbf{F}_q)}{\partial \mathbf{x}_{q_2}.y} \middle| \frac{\partial \text{vec}(\mathbf{F}_q)}{\partial \mathbf{x}_{q_3}.x} \middle| \frac{\partial \text{vec}(\mathbf{F}_q)}{\partial \mathbf{x}_{q_3}.y} \right] \\ &= \left[\begin{array}{cccccc} \frac{\partial f_{q11}}{\partial \mathbf{x}_{q_1}.x} & \frac{\partial f_{q11}}{\partial \mathbf{x}_{q_1}.y} & \frac{\partial f_{q11}}{\partial \mathbf{x}_{q_2}.x} & \frac{\partial f_{q11}}{\partial \mathbf{x}_{q_2}.y} & \frac{\partial f_{q11}}{\partial \mathbf{x}_{q_3}.x} & \frac{\partial f_{q11}}{\partial \mathbf{x}_{q_3}.y} \\ \frac{\partial f_{q21}}{\partial \mathbf{x}_{q_1}.x} & \frac{\partial f_{q21}}{\partial \mathbf{x}_{q_1}.y} & \frac{\partial f_{q21}}{\partial \mathbf{x}_{q_2}.x} & \frac{\partial f_{q21}}{\partial \mathbf{x}_{q_2}.y} & \frac{\partial f_{q21}}{\partial \mathbf{x}_{q_3}.x} & \frac{\partial f_{q21}}{\partial \mathbf{x}_{q_3}.y} \\ \frac{\partial f_{q12}}{\partial \mathbf{x}_{q_1}.x} & \frac{\partial f_{q12}}{\partial \mathbf{x}_{q_1}.y} & \frac{\partial f_{q12}}{\partial \mathbf{x}_{q_2}.x} & \frac{\partial f_{q12}}{\partial \mathbf{x}_{q_2}.y} & \frac{\partial f_{q12}}{\partial \mathbf{x}_{q_3}.x} & \frac{\partial f_{q12}}{\partial \mathbf{x}_{q_3}.y} \\ \frac{\partial f_{q22}}{\partial \mathbf{x}_{q_1}.x} & \frac{\partial f_{q22}}{\partial \mathbf{x}_{q_1}.y} & \frac{\partial f_{q22}}{\partial \mathbf{x}_{q_2}.x} & \frac{\partial f_{q22}}{\partial \mathbf{x}_{q_2}.y} & \frac{\partial f_{q22}}{\partial \mathbf{x}_{q_3}.x} & \frac{\partial f_{q22}}{\partial \mathbf{x}_{q_3}.y} \end{array} \right]_{4 \times 6}. \end{aligned} \quad (1.15)$$

1.5.2.3 计算 $\partial \Psi_q / \partial \mathbf{f}_q$

能量函数 Ψ_q 可以使用从变形梯度 \mathbf{F}_q 导出的旋转不变量 $\{I_1, I_2, I_3\}$ 表示，所以有

$$\frac{\partial \Psi_q}{\partial \mathbf{f}_q} = \sum_i \frac{\partial \Psi_q}{\partial I_i} \frac{\partial I_i}{\partial \mathbf{f}_q}. \quad (1.16)$$

对于上式中的 $\partial \Psi_q / \partial I_i$ ，使用不变量改写对称迪利克雷能量的二维形式 $\Psi_{SD}^{2D} = \|\mathbf{F}\|^2 + \|\mathbf{F}^{-1}\|^2$ 为

$$\Psi_q = \Psi_{SD}^{2D} = \frac{1}{2} \left(I_2 + \frac{I_2}{I_3^2} \right), \quad (1.17)$$

令 $\ell^\top = [I_1, I_2, I_3]^\top$ ，有

$$\frac{\partial \Psi_{SD}^{2D}}{\partial \ell^\top} = \begin{bmatrix} \frac{\partial \Psi_{SD}^{2D}}{\partial I_1} & \frac{\partial \Psi_{SD}^{2D}}{\partial I_2} & \frac{\partial \Psi_{SD}^{2D}}{\partial I_3} \end{bmatrix}^\top = \begin{bmatrix} 0 & \frac{1}{2} \left(1 + \frac{1}{I_3^2} \right) & -\frac{I_2}{I_3^3} \end{bmatrix}^\top, \quad (1.18)$$

对于 $\partial I_i / \partial \mathbf{f}_q$ 有

$$\begin{cases} \frac{\partial I_1}{\partial \mathbf{f}_q} = \text{vec} \left(\frac{\partial I_1}{\partial \mathbf{F}} \right) = \text{vec}(\mathbf{R}) = \mathbf{r}, \\ \frac{\partial I_2}{\partial \mathbf{f}_q} = \frac{\partial \|\mathbf{S}\|^2}{\partial \mathbf{f}_q} = \frac{\partial \|\mathbf{F}_q\|^2}{\partial \mathbf{f}_q} = \frac{\partial \|\mathbf{f}_q\|^2}{\partial \mathbf{f}_q^2} = 2\mathbf{f}, \\ \frac{\partial I_3}{\partial \mathbf{f}_q} = \mathbf{g} = \text{vec}(\mathbf{G}) = \text{vec} \left(\mathbf{U} \begin{bmatrix} \sigma_2 & 0 \\ 0 & \sigma_1 \end{bmatrix} \mathbf{V}^\top \right). \end{cases} \quad (1.19)$$

1.5.2.4 计算 $\partial^2 \Psi_q / \partial \mathbf{f}_q^2$

此章节我们略去对 $\partial^2 \Psi_q / \partial \mathbf{x}_q^2$ (即“海森矩阵”) 的特征值和特征向量的详细推导。 $\partial^2 \Psi_q / \partial \mathbf{f}_q^2$ 为

$$\begin{aligned} \frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} &= \sum_i \frac{\partial \Psi_q}{\partial I_i} \frac{\partial^2 \mathbf{I}_i}{\partial \mathbf{f}_q^2} \\ &\quad + \sum_i \frac{\partial^2 \Psi_q}{\partial I_i^2} \left(\frac{\partial \mathbf{I}_i}{\partial \mathbf{f}_q} \right) \left(\frac{\partial \mathbf{I}_i}{\partial \mathbf{f}_q} \right)^\top \\ &\quad + \sum_{i < j} \frac{\partial^2 \Psi_q}{\partial I_i \partial I_j} \left[\left(\frac{\partial \mathbf{I}_i}{\partial \mathbf{f}_q} \right) \left(\frac{\partial \mathbf{I}_j}{\partial \mathbf{f}_q} \right)^\top + \left(\frac{\partial \mathbf{I}_j}{\partial \mathbf{f}_q} \right) \left(\frac{\partial \mathbf{I}_i}{\partial \mathbf{f}_q} \right)^\top \right]. \end{aligned} \quad (1.20)$$

上式中用红色和蓝色标出的二阶张量都是由不变量对向量化后的形变梯度 \mathbf{f}_q 的偏导数组成的。通过分析不变量的特征系统，我们可以求出 $\partial^2 \Psi_q / \partial \mathbf{x}_q^2$ 的特征系统如下：

$$\begin{aligned} \lambda_1^{2D} &= 1 + \frac{3}{\sigma_1^4}, & \lambda_2^{2D} &= 1 + \frac{3}{\sigma_2^4}, & \lambda_3^{2D} &= 1 + \frac{1}{I_3^2} + \frac{I_2}{I_3^3}, & \lambda_4^{2D} &= 1 + \frac{1}{I_3^2} - \frac{I_2}{I_3^3}, \\ \mathbf{e}_1 &= \mathbf{d}_1, & \mathbf{e}_2 &= \mathbf{d}_2, & \mathbf{e}_3 &= \mathbf{l}, & \mathbf{e}_4 &= \mathbf{t}. \end{aligned} \quad (1.21)$$

其中

$$\begin{aligned} \mathbf{d}_1 &= \text{vec}(\mathbf{D}_1) = \text{vec}\left(\mathbf{U} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{V}^\top\right), & \mathbf{d}_2 &= \text{vec}(\mathbf{D}_2) = \text{vec}\left(\mathbf{U} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{V}^\top\right), \\ \mathbf{l} &= \text{vec}(\mathbf{L}) = \text{vec}\left(\frac{1}{\sqrt{2}} \mathbf{U} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathbf{V}^\top\right), & \mathbf{t} &= \text{vec}(\mathbf{T}) = \text{vec}\left(\frac{1}{\sqrt{2}} \mathbf{U} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{V}^\top\right). \end{aligned}$$

矩阵 \mathbf{L} 和 \mathbf{T} 分别称为翻转 (flip) 矩阵和扭曲 (twist) 矩阵。为将 $\partial^2 \Psi_q / \partial \mathbf{x}_q^2$ 投影为半正定矩阵，使用

$$\frac{\partial^2 \Psi_q}{\partial \mathbf{x}_q^2} \leftarrow \sum_{i=1}^4 \max(\lambda_i^{2D}, 0) \mathbf{e}_i \mathbf{e}_i^\top, \quad (1.22)$$

代替 $\partial^2 \Psi_q / \partial \mathbf{x}_q^2$ 的原始值。上式表示矩阵 $\partial^2 \Psi_q / \partial \mathbf{x}_q^2$ 被投影为半正定矩阵 (矩阵的特征值不小于 0)。根据公式 1.10b 和公式 1.12b， $\partial^2 \Psi / \partial \mathbf{x}^2$ 的半正定性得以保证。实际上，根据

$$\frac{\partial^2 \Psi_{SD}^{2D}}{\partial \ell^\top \partial \ell} = \begin{bmatrix} \frac{\partial^2 \Psi_{SD}^{2D}}{\partial^2 I_1} & \frac{\partial^2 \Psi_{SD}^{2D}}{\partial I_1 \partial I_2} & \frac{\partial^2 \Psi_{SD}^{2D}}{\partial I_1 \partial I_3} \\ \frac{\partial^2 \Psi_{SD}^{2D}}{\partial I_2 \partial I_1} & \frac{\partial^2 \Psi_{SD}^{2D}}{\partial^2 I_2} & \frac{\partial^2 \Psi_{SD}^{2D}}{\partial I_2 \partial I_3} \\ \frac{\partial^2 \Psi_{SD}^{2D}}{\partial I_3 \partial I_1} & \frac{\partial^2 \Psi_{SD}^{2D}}{\partial I_3 \partial I_2} & \frac{\partial^2 \Psi_{SD}^{2D}}{\partial^2 I_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{I_3^3} \\ 0 & -\frac{1}{I_3^3} & \frac{3I_3}{I_3^4} \end{bmatrix},$$

和文章 [2] 在 5.5 节中的介绍，我们可以与公式 1.20 相对应的 $\partial^2 \Psi_q / \partial \mathbf{x}_q^2$ 的展开形式：

$$\frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} = \left(1 + \frac{1}{I_3^2}\right) \mathbf{I} - \frac{I_2}{I_3^3} (\hat{\mathbf{r}}\hat{\mathbf{r}}^\top + \mathbf{t}\mathbf{t}^\top - \mathbf{p}\mathbf{p}^\top - \mathbf{l}\mathbf{l}^\top) - \frac{2}{I_3^3} (\mathbf{g}\mathbf{f}^\top + \mathbf{f}\mathbf{g}^\top).$$

其中

$$\hat{\mathbf{r}} = \frac{\mathbf{r}}{\sqrt{2}}, \quad \mathbf{p} = \text{vec}(\mathbf{P}) = \text{vec}\left(\frac{1}{\sqrt{2}} \mathbf{U} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \mathbf{V}^\top\right).$$

矩阵 \mathbf{P} 被称为挤压 (pinch) 矩阵。可以验证如下表达式成立：

$$\left(1 + \frac{1}{I_3^2}\right) \mathbf{I} - \frac{I_2}{I_3^3} (\hat{\mathbf{r}}\hat{\mathbf{r}}^\top + \mathbf{t}\mathbf{t}^\top - \mathbf{p}\mathbf{p}^\top - \mathbf{l}\mathbf{l}^\top) - \frac{2}{I_3^3} (\mathbf{g}\mathbf{f}^\top + \mathbf{f}\mathbf{g}^\top) = \sum_{i=1}^4 \lambda_i^{2D} \mathbf{e}_i \mathbf{e}_i^\top.$$

1.6 算法流程

我们在 1.4.1 中提到，牛顿法往往用于优化粗略的、低精度的解。我们使用 Tutte’s Embedding 作为无翻转的、满足条件的低精度解（算法 1），并使用牛顿法对其进行优化，使得能量函数达到极小。

Algorithm 1: Tutte Embedding 算法流程

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

Output: $\mathcal{P} = \mathcal{P}(G, U_b)$, $U_b = \{P_1, P_2, \dots, P_{N_V}\}$

- 1 初始化系数矩阵 \mathbf{A} 为单位矩阵;
 - 2 计算 S 的边界 ∂G , 根据结果对结点重新编号;
 - 3 将正 n 边形的顶点坐标赋予 $\mathbf{b}_{N_V \times 2} = [P_1, P_2, \dots, P_n, 0, \dots, 0]^T$;
 - 4 **for** $k = n + 1$ **to** N_V **do**
 - 5 计算 Tutte 权重或者 Floater 权重 λ_{k,j_l} , $l \in \{1, 2, \dots, d_k\}$;
 - 6 将权重 λ_{k,j_l} 赋予矩阵 \mathbf{A} ;
 - 7 **end**
 - 8 求解方程组 $\mathbf{Ax} = \mathbf{b}$, 得到 $\mathbf{x}_{N_V \times 2} = [P_1, P_2, \dots, P_n, P_{n+1}, \dots, P_{N_V}]^T$;
 - 9 输出 $\mathcal{P} = \mathcal{P}(G, U_b)$;

在介绍算法之前，我们需要明确输入三角形网格 $S = (G, \mathbf{X})$ 与迭代步过程中的平面参数化网格 $\mathcal{P} = \mathcal{P}(G, U_b)$ 的不同。文章 [1] 的最优化方法，是对输入三角形网格模型的平面参数化结果 $\mathcal{P}(G, U_b)$ 中的顶点位置集合 U_b 进行优化的方法。所以在优化过程中，集合 \mathbf{X} 内的点的位置 $\bar{\mathbf{x}}$ 不会发生改变。每一次迭代步都要计算能量函数 $\Psi(\mathbf{x}^k)$ 的牛顿方向 \mathbf{d}^k 并更新 U_b 中的顶点位置。

1.6.1 投影牛顿法

基于之前章节的内容，我们在此给出文章 [1] 中的方法，也是“投影牛顿法”的算法流程。

Algorithm 2: 投影牛顿法 (Project Newton Solver)

Input: 三角形曲面网格 $S(G, \mathbf{X})$ 的 Tutte's Embedding 参数化结果 $\mathcal{P}(G, U_b)$ 与最大迭代次数 M 。

Output: 基于能量函数 Ψ_{SD}^{2D} 的参数化优化结果。

/* 为了统一符号, 使用 x^k 代表上一次迭代步的结果, x^{k+1} 表示这一次更新后的结果. */

- ```

1 初始化 \mathbf{x}^0 ;
2 for $i \leftarrow 0$ to M do
3 $\mathbf{b}^k \leftarrow \nabla \Psi_{\text{SD}}^{\text{2D}}(\mathbf{x}^k)$; /* 公式 1.10a */
4 if $\|\mathbf{b}^k\| \leq 10^{-4}$ then /* 这里使用无穷范数, 使用 2 范数也可以 */
5 | return \mathbf{x}^k ;
6 end
7 $\mathbf{H}^k \leftarrow \text{Project_Hessian}(\mathbf{x}^k)$; /* 计算半正定投影海森矩阵, 算法 3 */
8 $\mathbf{d}^k \leftarrow -[\mathbf{H}^k]^{-1}\mathbf{b}^k$; /* 牛顿方向, 等价于求解 $\mathbf{H}^k \mathbf{d}^k = \mathbf{b}^k$, LDLT 分解法 + shift */
9 $\alpha_i \leftarrow \text{Line_Search}(\mathbf{x}^k, \mathbf{b}^k, \mathbf{d}^k)$; /* 对步长 α^k 进行搜索, 算法 4 */
10 $\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \alpha^k \mathbf{d}^k$;
11 end
12 return \mathbf{x}^M ;

```

注意算法 2 中第 1 行的初始化操作。 $\mathbf{x}^0$  并不是三角网格的参数化结果中集合  $U_b$  内的顶点坐标，而是  $\mathbf{X}$  中的顶点  $\bar{\mathbf{x}}$  在三角形内的局部坐标。我们需要建立局部坐标系，并求得三角形的顶点在局部坐标系下的坐标。如图 1.3，对于处在三维空间的三角形  $T$  的顶点  $\bar{\mathbf{x}}_i \in \mathbb{R}^3$ ，在右图所示的局部坐标系下，有

$$\begin{cases} \bar{\mathbf{x}}_1 \bar{\mathbf{x}}_2 = (\|\bar{\mathbf{x}}_1 \bar{\mathbf{x}}_2\|, 0), \\ \bar{\mathbf{x}}_1 \bar{\mathbf{x}}_3 = \frac{1}{\|\bar{\mathbf{x}}_1 \bar{\mathbf{x}}_2\|} (\bar{\mathbf{x}}_1 \bar{\mathbf{x}}_2 \cdot \bar{\mathbf{x}}_1 \bar{\mathbf{x}}_3, \det([\bar{\mathbf{x}}_1 \bar{\mathbf{x}}_2 \quad \bar{\mathbf{x}}_1 \bar{\mathbf{x}}_3])) . \end{cases}$$

此处对  $\mathbf{x}^0$  的初始化方法并不唯一，但是不可以使用  $U_b$  内的顶点坐标进行初始化。否则算法 2 计算得到的  $\mathbf{b}^0$  是零向量，算法会提前终止，无法计算牛顿方向  $\mathbf{d}^0$  并更新  $\mathbf{x}^0$ 。

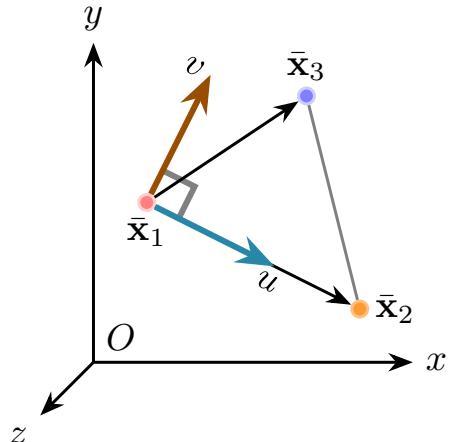


图 1.3: 三角形局部坐标系

### Algorithm 3: 海森矩阵投影 (Project Hessian)

**Input:** 顶点  $\mathbf{x}^k$ 。

**Output:** 投影后的海森矩阵  $\mathbf{H}$ 。

```

1 $\mathbf{H} \leftarrow \mathbf{0};$
2 for 每个基点 q do /* 等价于遍历每个三角面 */
3 $\mathbf{F}_q \leftarrow \text{Compute_Deformation_Gradient}(q, \mathbf{x}^k);$ /* 公式 1.15 */
4 $\mathbf{f}_q \leftarrow \text{vec}(\mathbf{F}_q);$
5 $\{\mathbf{U}, \Sigma, \mathbf{V}\} \leftarrow \text{Compute_SVD}(\mathbf{F}_q);$ /* 章节 1.5, 使用 Eigen::JacobiSVD 并矫正 */
6 $\{\lambda_i, \mathbf{e}_i\}_{i=1}^4 \leftarrow \text{Eval_Energy_EigenSystem}(\mathbf{U}, \Sigma, \mathbf{V});$ /* 章节 1.5.2.4 */
7 $\mathbf{H}_q \leftarrow \sum_i \max(\lambda_i, 0) \mathbf{e}_i \mathbf{e}_i^\top;$
8 $\mathbf{H} \leftarrow \mathbf{H} + |q| (\partial \mathbf{f}_q / \partial \mathbf{x}_q)^\top \mathbf{H}_q (\partial \mathbf{f}_q / \partial \mathbf{x}_q);$ /* 公式 1.10b, 章节 1.5.2.1 */
9 end
10 return $\mathbf{H};$
```

## 1.6.2 线搜索

算法 2 第 9 行 Line\_Search 函数查找在  $\mathbf{x}^k$ 、 $\mathbf{b}^k$  和  $\mathbf{d}^k$  下的合适步长，属于“线搜索类算法”：

### 定义 1.3 (线搜索类算法 [6])

给定当前迭代点  $\mathbf{x}^k$ ，首先通过某种算法选取向量  $\mathbf{d}^k$ ，之后确定正数  $\alpha_k$ ，则下一步迭代点可写作

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k.$$

称  $\mathbf{d}^k$  为迭代点  $\mathbf{x}^k$  处的搜索方向， $\alpha_k$  为相应的步长。这里要求  $\mathbf{d}^k$  是一个下降方向，即  $(\mathbf{d}^k)^\top \nabla f(\mathbf{x}^k) < 0$ 。这个下降性质保证了沿着此方向搜索函数  $f$  的值会减小。此类算法的关键是如何选取一个好的方向  $\mathbf{d}^k \in \mathbb{R}^d$  以及合适的步长  $\alpha^k$ 。



选取  $\alpha^k$  需要满足一定的要求，称为线搜索准则。常见的方法 [6] 有 Armijo 准则、Goldstein 准则、Wolfe 准则和非单调线性搜索准则等。线搜索方法直接影响到投影牛顿法的收敛速度，选取合适的线搜索方法十分重要。文章 [1] 中称对于称迪利克雷能量，使用满足 Armijo 准则的“回退线搜索”方法查找  $\alpha^k$  更合适。我们这里给出 Line\_Search 方法的回退线搜索版本 [6]：

**Algorithm 4:** 回退线搜索方法 (Backtracking Line Search)

---

**Input:** 迭代点  $\mathbf{x}^k$ , 牛顿方向  $\mathbf{d}^k$ , 最大迭代次数  $M_L$ , 收缩系数  $\gamma$ , 控制系数  $c$ 。

**Output:** 满足条件的步长  $\alpha^k$ 。

```

1 初始化 $\hat{\alpha}^k$; /* 满足三角形无翻转的初始化策略 */
2 for $i \leftarrow 0$ to M_L do /* 固定迭代次数防止 $\hat{\alpha}^k$ 过小 */
3 if $\Psi_{\text{SD}}^{2D}(\mathbf{x}^k + \hat{\alpha}^k \mathbf{d}^k) \leq \Psi_{\text{SD}}^{2D}(\mathbf{x}^k) + c\hat{\alpha}^k \nabla \Psi_{\text{SD}}^{2D}(\mathbf{x}^k)^\top \mathbf{d}^k$ then /* 满足 Armijo 准则 [6] */
4 break;
5 end
6 $\hat{\alpha}^k \leftarrow \gamma \hat{\alpha}^k$;
7 end
8 return $\alpha^k = \hat{\alpha}^k$;

```

---

### 1.6.3 保证无翻转的最大步长

算法 4 中初始化  $\hat{\alpha}^k$  时需要注意, 牛顿方向  $\mathbf{d}^k$  虽然会让能量函数  $\Psi_{\text{SD}}^{2D}$  减小, 但会让  $\mathbf{x}^k + \alpha^k \mathbf{d}^k$  的三角形出现翻转和退化。在迭代的过程中, 翻转和退化的三角形会影响后续的迭代过程, 最终得到存在少量翻转的参数化结果。为解决这个问题, 考虑文章 [9, 11] 中的无翻转线搜索方法, 每次搜索之前需要确定迭代步  $\mathbf{x}^k$  在  $\mathbf{d}^k$  下得到无翻转结果的最小步长  $\alpha_{\text{upper}}^k$ , 并选取  $\hat{\alpha}^k = \min(0.99\alpha_{\text{upper}}^k, 1.0)$ 。

随着步长  $\alpha^k$  从 0 增长到 1, 一些三角形会经历无翻转  $\rightarrow$  退化为直线、点  $\rightarrow$  发生翻转的过程。我们要计算让三角形出现退化的所有  $\alpha_q^k$  中的最小值。考虑迭代步  $\mathbf{x}_q^k$ 。三角形  $T_q^k$  基于牛顿方向  $\mathbf{d}_q^k$  和步长  $\alpha_q^k$  更新后的三角形为  $T_q^{k+1}$ 。此过程中三角形的顶点坐标满足

$$\mathbf{F}_q^k \mathbf{D}_m = \underbrace{\begin{bmatrix} \mathbf{x}_{q2}^k - \mathbf{x}_{q1}^k & | \\ \mathbf{x}_{q3}^k - \mathbf{x}_{q1}^k & \end{bmatrix}}_{\mathbf{D}_s^k} + \alpha_q^k \underbrace{\begin{bmatrix} \mathbf{d}_{q2}^k - \mathbf{d}_{q1}^k & | \\ \mathbf{d}_{q3}^k - \mathbf{d}_1^k & \end{bmatrix}}_{\mathbf{D}^k} = \underbrace{\begin{bmatrix} \mathbf{x}_{q2}^{k+1} - \mathbf{x}_{q1}^{k+1} & | \\ \mathbf{x}_{q3}^{k+1} - \mathbf{x}_{q1}^{k+1} & \end{bmatrix}}_{\mathbf{D}_s^{k+1}}.$$

若三角形发生退化, 说明  $|\mathbf{D}_s^{k+1}| = |\mathbf{D}_s^k + \alpha_q^k \mathbf{D}^k| = 0$ , 按分量展开并计算可得

$$|\mathbf{D}^k|(\alpha_q^k)^2 + (|\mathbf{D}_s^k| + |\mathbf{D}^k| - |\mathbf{D}_s^k - \mathbf{D}^k|)\alpha_q^k + |\mathbf{D}_s^k| = 0. \quad (1.23)$$

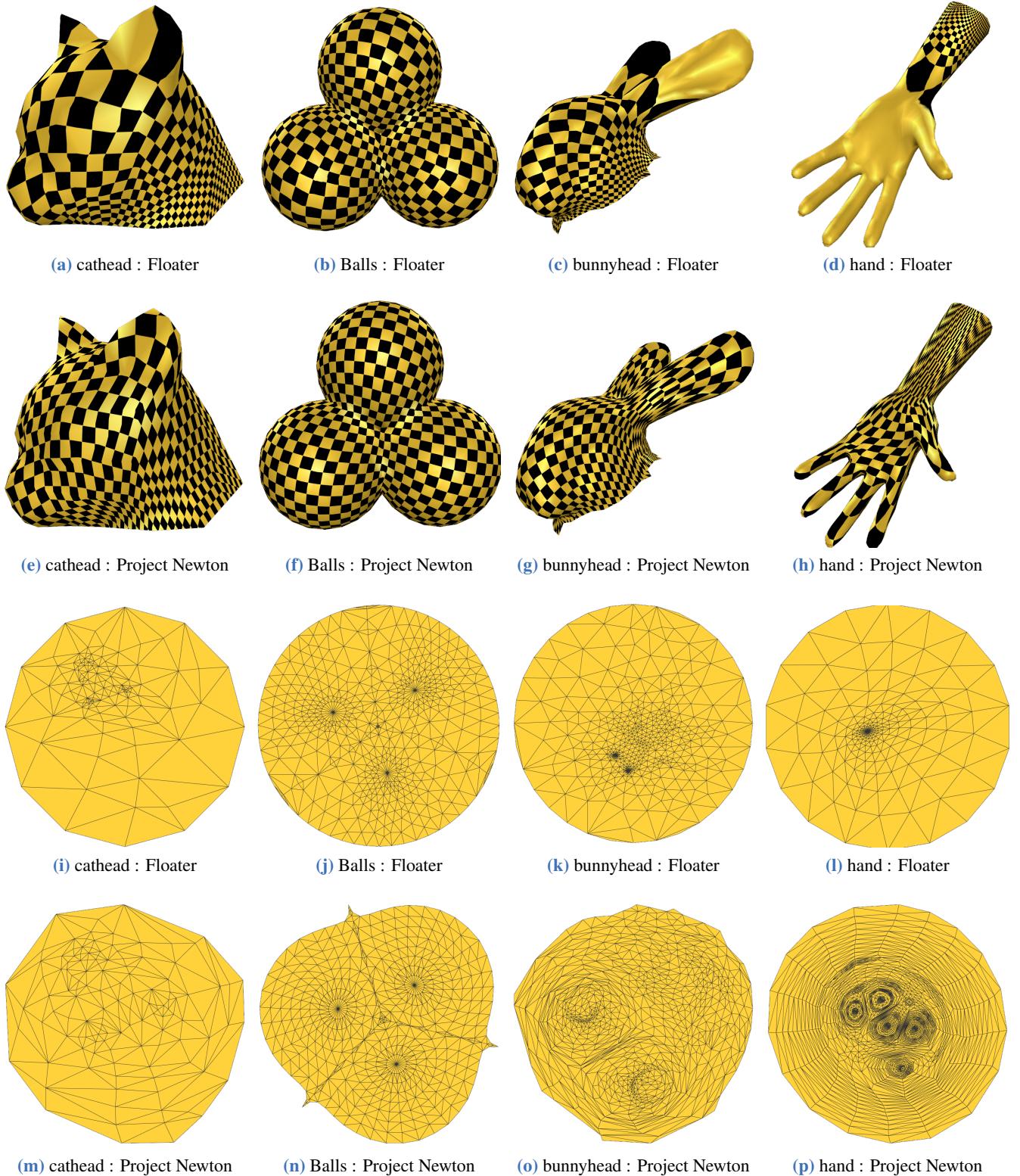
求解后得到的较小根是发生退化的  $\alpha_q^k$  上界。 $\alpha_{\text{upper}}^k$  是所有三角形的  $\alpha_q^k$  的最小值  $\alpha_{\text{upper}}^k = \min_q \alpha_q^k$ 。

## 1.7 实验结果

我们使用四个模型测试本文实现的有效性 (表 1.3)。

| 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 |

表 1.3: 测试数据



**图 1.4:** 与 Tutte’s Embedding 方法的对比。第一排为基于 Floater’s Weight 纹理映射贴图绘制结果，第二排为对应的投影牛顿法优化的纹理映射贴图绘制结果；第三排为基于 Floater’s Weight 平面参数化结果，第四排为对应的投影牛顿法优化的平面参数化结果。初始 Tutte’s Embedding 的边界形状为正多边形。

从图 1.4 可以看出，相较于 Tutte’s Embedding，投影牛顿法的优势体现在：

1. 投影牛顿法在 Tutte’s Embedding 方法结果中纹理坐标高扭曲的区域（如图 1.4(c) 中的兔耳朵区域

和图 1.4(d) 中的手掌、手指区域) 有着更好的表现。

2. 投影牛顿法得到的结果中棋盘格的大小相近。即使图 1.4(g) 的兔耳根部和图 1.4(h) 的手臂处棋盘格扭曲严重, 但格子尺寸差异不大。
3. 投影牛顿法可以保证低扭曲区域纹理映射的平直。Floater's Weight 的参数化结果 (如图 1.4(b) 高光处) 中纹理映射出现了扭曲。但投影牛顿法的优化结果 (和图 1.4(g) 高光处) 更为平直。

基于能量函数优化过程, 是保持输入三角网格的三角形性质的过程。Tutte's Embedding 的结果具有“与边界距离越远的顶点越密集, 相邻的三角面变形越大”的特点, 严重变形的三角面是投影牛顿法优化的主要对象。投影牛顿法可以将变形严重的三角面 (距离边界太远的顶点周围) 拉伸展平, 使得纹理坐标映射更均匀。但是代价是“特殊位置”的三角形出现严重变形, 如图 1.4(g) 中的兔耳根部和图 1.4(h) 中的手臂处, 棋盘格严重的扭曲变形。此现象也可以在图 1.4(o) 和图 1.4(p) 中看出。

另外, 我们罗列了投影牛顿法的性能表现 (表 1.4)。

| 名称        | 迭代次数 | 能量下降范围                                     | 平面参数化时间   | 投影牛顿法时间    |
|-----------|------|--------------------------------------------|-----------|------------|
| cathead   | 19   | 369.152 → 10.0101                          | 1.012 ms  | 29.938 ms  |
| Balls     | 58   | 229232 → 769.848                           | 6.275 ms  | 256.02 ms  |
| bunnyhead | 33   | 0.9461 → 0.0695                            | 9.357 ms  | 221.13 ms  |
| hand      | 509  | $4.0366 \times 10^{18} \rightarrow 277247$ | 31.661 ms | 5549.44 ms |

表 1.4: 投影牛顿法的性能表现

表 1.4 中的迭代次数和投影牛顿法用时基本上随着模型的顶点规模增加而增加。bunnyhead 模型的全局能量函数范围比 cathead 和 Balls 小。原因是 bunnyhead 的包围盒较小, 三角形的变形也较小。bunnyhead 的投影牛顿法的迭代次数较少, 计算用时小于 cathead 和 Balls, 但原因未知。可能海森矩阵在大部分迭代步  $\mathbf{x}^k$  处正定, 收敛速度较快。

最后, 我们尝试对中等规模的模型 (hilbert.obj) 进行测试, 测试结果如表 1.5, 绘制结果如图 1.5。

| 数据信息                                                         | 迭代次数 | 能量下降范围                                       | 平面参数化时间  | 投影牛顿法时间   |
|--------------------------------------------------------------|------|----------------------------------------------|----------|-----------|
| $(V, E, F) = (79729, 226896, 147168)$<br>边界长度 12288, 5.18 MB | 3968 | $2.03854 \times 10^{11} \rightarrow 1025.54$ | 164.18 s | 28 m 43 s |

表 1.5: 测试数据: hilbert.obj

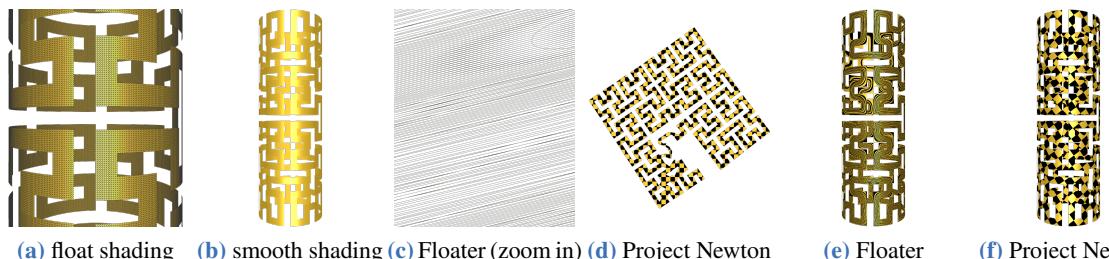


图 1.5: hilbert.obj 在收敛条件为  $\|\mathbf{b}^k\| \leq 10^{-5}$  下的测试结果 (图 (a) 和图 (b) 是三维模型渲染效果, 图 (c) 和图 (d) 是参数化优化前后效果, 图 (e) 和图 (f) 是对应的纹理映射贴图。边界形状: 正多边形)

从图 1.5 中看到, 投影牛顿法对于中等规模的数据也有较好的表现。但是表 1.5 中计算时间和迭代次数极大地增加, 想要减少计算时间, 还需要对方法进行额外的优化。

**注** 我们省略了不同边界形状对结果的影响。此外，实际在测试过程中，基于 Tutte's Weight 和基于 Floater's Weight 的平面参数化初值不会对投影牛顿法的结果造成显著的影响。

## 1.8 总结

本次作业实现了文章 [1] 提出的基于特征系统的各向异性能量的曲面参数化优化方法，优化结果在纹理坐标映射和运行性能方面都十分优秀。但是方法较为复杂，实现难度较大，且文章的不少细节处需要仔细辨别，如果理解不清会导致实现结果出现翻转、错位、坐标爆炸等等问题。本文的实现过程参考了重要的配套讲义 [2]。在此称赞 Pixar 的研究学者，为我们呈现了如此精彩出色的工作。

## 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 // ===== hw2: Actions =====
5 QAction* actProjNewtonSolver;
6 // ...
7 };
8
9 // ===== surfacemeshprocessing.cpp =====
10 void SurfaceMeshProcessing::CreateActions(void)
11 {
12 // ...
13 // ===== hw2: Actions =====
14 actProjNewtonSolver = new QAction("Project Newton Solver", this);
15 actProjNewtonSolver->setStatusTip(tr("Optimize Tutte's Embedding with Project-Newton-
16 Solver"));
17 connect(actProjNewtonSolver, SIGNAL(triggered()), viewer, SLOT(ProjNewtonSolver()));
18 }
```

Listing A.2: MainViewerWidget.\*

```
1 // ===== MainViewerWidget.h =====
2 class MainViewerWidget : public QDialog
3 {
4 // ...
5 // ===== hw2: Project Newton Solver =====
6 void ProjNewtonSolver();
7 // ...
8 };
9
10 // ===== MainViewerWidget.cpp =====
11 // ...
12 // ===== hw2: Project Newton Solver API =====
13 void MainViewerWidget::ProjNewtonSolver()
14 {
15 meshviewerwidget->ProjNewtonSolver();
16 }
17 // ...
```

Listing A.3: MeshViewerWidget.\*

```

1 // ===== MeshViewerWidget.h =====
2 class MeshViewerWidget : public QGLViewerWidget
3 {
4 // ...
5 // ===== hw2: Project Newton Method =====
6 void ProjNewtonSolver();
7 // ...
8 // ===== hw2: Project Newton Method member =====
9 bool isParameterized = false;
10 bool isProjNewtonSolver = false;
11
12 Eigen::SparseMatrix<double> paramUVs;
13
14 eigensys::ProjectNewtonSolver m_ProjNewtonSolver;
15 };
16
17 // ===== MeshViewerWidget.cpp =====
18 // ...
19 // ===== hw2: Project Newton Solver API =====
20 void MeshViewerWidget::ProjNewtonSolver()
21 {
22 if (polyMesh->numVertices() == 0)
23 {
24 std::cerr << "ERROR: ProjNewtonSolver() No vertices!" << std::endl;
25 return;
26 }
27
28 if (!isParameterized)
29 {
30 std::cout << "The mesh hasn't been parameterized yet, run with Tutte's Embedding
31 ... \n";
32 paramUVs = this->TutteParam(TutteParamType::FLOATER_WEIGHTED);
33 isParameterized = true;
34 }
35
36 if (!isProjNewtonSolver)
37 {
38 m_ProjNewtonSolver.PresetMeshUV(polyMesh, paramUVs);
39 isProjNewtonSolver = true;
40 }
41
42 auto timeStart = std::chrono::steady_clock::now();
43
44 while (m_ProjNewtonSolver.UpdateMeshUV(polyMesh));
45
46 auto timeEnd = std::chrono::steady_clock::now();
47 auto ms = std::chrono::duration_cast<std::chrono::microseconds>(timeEnd - timeStart).

```

```

 count() / 1000.0;

47
48 std::cout << "Project Newton Solver finished with " << ms << " ms\n";
49
50 m_ProjNewtonSolver.SaveEnergies("energies.txt");
51
52 isProjNewtonSolver = false;
53
54 UpdateMesh();
55 update();
56 }
57 // ...

```

## A.2 新增文件

**文件** 在相关代码存放在目录 Surface\_Framework\_Cmake/src/homeworks/EnergyEigensystems/:

- (a) Util\_DataStructure.h
- (b) Util\_DataStructure.cpp
- (c) Util\_EnergyEigensystems.h
- (d) Util\_EnergyEigensystems.cpp

**Listing A.4:** Util\_DataStructure.h

```

1 // ===== Util_DataStructure.h =====
2 #pragma once
3
4 #include <array>
5 #include <Eigen\.Dense>
6 #include "../PolyMesh/include/PolyMesh/PolyMesh.h"
7
8 namespace eigensys
9 {
10 inline Eigen::Vector4d opVEC(const Eigen::Matrix2d& M)
11 {
12 return Eigen::Vector4d{ M(0, 0), M(1, 0), M(0, 1), M(1, 1) };
13 }
14
15 // return vertex->index(), multiply it by 2 for UV
16 // We use idxs as indexes, which refers to the index of vertice in mesh->vertices()
17 // and use ids as indices, which refers to the index of UV, and indices = indexes
18 * 2
19 inline Eigen::Vector3i PolyFaceVertIdxs(acamcad::polymesh::PolyMesh* mesh, size_t
20 faceID)
21 {
22 auto verts = mesh->polygonVertices(mesh->polyface(faceID));
23 return Eigen::Vector3i{ verts[0]->index(), verts[1]->index(), verts[2]->index() };
24 }

```

```

23
24 struct QPW_Invariables
25 {
26 void CalculateInvariants(const Eigen::Matrix2d& S);
27 double I1{ 2.0 }; // I1 = sum(sigma_i)
28 double I2{ 2.0 }; // I2 = sum(sigma_i^2)
29 double I3{ 1.0 }; // I3 = prod(sigma_i)
30 };
31
32 struct QPW_Decomposition
33 {
34 void CalculateSVDnPolar(const Eigen::Matrix2d& F);
35 Eigen::Vector2d sigma; // { U, Sigma, V } = SVD(F)
36 Eigen::Matrix2d U, V;
37 Eigen::Matrix2d S, R; // polar decomposition
38 };
39
40 struct QPW_DeformVectors
41 {
42 QPW_DeformVectors(const Eigen::Matrix2d& F, const QPW_Decomposition& decomp);
43
44 Eigen::Vector4d f{ Eigen::Vector4d::Zero() }; // vec(F)
45 Eigen::Vector4d g{ Eigen::Vector4d::Zero() }; // vec(R)
46 Eigen::Vector4d r{ Eigen::Vector4d::Zero() }; // r / sqrt(2)
47 Eigen::Vector4d t{ Eigen::Vector4d::Zero() }; // vec(U twist VT) / sqrt(2)
48 Eigen::Vector4d p{ Eigen::Vector4d::Zero() }; // vec(U [1 -1] VT) / sqrt(2)
49 Eigen::Vector4d l{ Eigen::Vector4d::Zero() }; // vec(U flip VT) / sqrt(2)
50 Eigen::Vector4d d1{ Eigen::Vector4d::Zero() }; // vec(U {1 0} VT)
51 Eigen::Vector4d d2{ Eigen::Vector4d::Zero() }; // vec(U {0 1} VT)
52 };
53
54 // cell, storing Dm and pfq_pfq
55 class QPW_Cell
56 {
57 friend struct QPW_EigenSystem2D;
58
59 public:
60 using vec6d = Eigen::Vector<double, 6>;
61 using mat6d = Eigen::Matrix<double, 6, 6>;
62
63 QPW_Cell(const Eigen::Matrix2d& dm);
64
65 double GetVolumeWeight() const { return Dm.determinant() / 2.0; }
66 Eigen::Matrix<double, 4, 6> GetDeformGradDerivative() const { return pfq_pfq; }
67
68 std::pair<vec6d, mat6d> CalculateGradNHess(const Eigen::Matrix2d& Ds);
69 double CalculateEnergy(const Eigen::Matrix2d& Ds);
70 double GetLastEnergy() const { return CalculateEnergy(m_Invariables); }
71

```

```

72 private:
73 double CalculateEnergy(const QPW_Invariables& invars) const;
74 QPW_DeformVectors CalculateVars(const Eigen::Matrix2d& F);
75 Eigen::Vector4d CalculateEnergyDeformGrad(const QPW_Invariables& invars, const
76 QPW_DeformVectors& defvecs) const;
77 Eigen::Matrix4d CalculateEnergyDeformHess(const QPW_EigenSystem2D& eigensys) const;
78
79 private:
80 Eigen::Matrix2d Dm; // F = Ds / Dm
81 Eigen::Matrix<double, 4, 6> pfq_pxq{ Eigen::Matrix<double, 4, 6>::Zero() }; // pf
82 // px per quad-point
83
84 };
85
86 // eigen system
87 struct QPW_EigenValVecPair
88 {
89 double l{ 0.0 };
90 Eigen::Vector4d e{ Eigen::Vector4d::Zero() };
91 };
92
93 struct QPW_EigenSystem2D
94 {
95 QPW_EigenSystem2D(const QPW_Cell& cell, const QPW_DeformVectors& defvecs);
96 std::array<QPW_EigenValVecPair, 4> pairs;
97 };
98 }
```

Listing A.5: Util\_DataStructure.cpp

```

1 // ===== Util_DataStructure.cpp =====
2 #include "Util_DataStructure.h"
3
4 #include <numeric>
5 #include <algorithm>
6
7 namespace eigensys
8 {
9 // member functions
10 void QPW_Invariables::CalculateInvariants(const Eigen::Matrix2d& S)
11 {
12 I1 = S.trace();
13 I2 = S.squaredNorm();
14 I3 = S.determinant();
15 }
16
17 void QPW_Decomposition::CalculateSVDnPolar(const Eigen::Matrix2d& F)
```

```

18 {
19 Eigen::JacobiSVD<Eigen::Matrix2d> SVD(F, Eigen::ComputeFullU | Eigen::ComputeFullV)
20 ;
21 SVD.computeU();
22 SVD.computeV();
23 U = SVD.matrixU();
24 V = SVD.matrixV();
25 sigma = SVD.singularValues();

26 Eigen::Matrix2d L = Eigen::Matrix2d::Identity();
27 if ((U * V.transpose()).determinant() < 0) L(1, 1) = -1;

28 double detU = U.determinant();
29 double detV = V.determinant();

30 if (detU < 0 && detV > 0) U = U * L;
31 if (detU > 0 && detV < 0) V = V * L;
32 sigma = (sigma.asDiagonal() * L).diagonal();

33 R = U * V.transpose();
34 S = V * sigma.asDiagonal() * V.transpose();
35 }

36 QPW_DeformVectors::QPW_DeformVectors(const Eigen::Matrix2d& F, const
37 QPW_Decomposition& decomp)
38 {
39 Eigen::Matrix2d twist;
40 twist << 0, -1, 1, 0;
41
42 Eigen::Matrix2d flip;
43 flip << 0, 1, 1, 0;
44
45 auto& U = decomp.U;
46 auto& sigma = decomp.sigma;
47 auto& VT = decomp.V.transpose();

48 double sqrt2 = std::sqrt(2.0);

49 //Eigen::Matrix2d G = twist * F * twist.transpose();
50 Eigen::Matrix2d G = U * sigma.reverse().asDiagonal() * VT;
51 Eigen::Matrix2d T = U * twist * VT / sqrt2;
52 Eigen::Matrix2d P = U * Eigen::Vector2d(1, -1).asDiagonal() * VT / sqrt2;
53 Eigen::Matrix2d L = U * flip * VT / sqrt2;
54 Eigen::Matrix2d D1 = U * Eigen::Vector2d(1, 0).asDiagonal() * VT;
55 Eigen::Matrix2d D2 = U * Eigen::Vector2d(0, 1).asDiagonal() * VT;

56 f = opVEC(F);
57 g = opVEC(G);
58 r = opVEC(decomp.R);

```

```

65 t = opVEC(T);
66 p = opVEC(P);
67 l = opVEC(L);
68 d1 = opVEC(D1);
69 d2 = opVEC(D2);
70 }
71
72 QPW_EigenSystem2D::QPW_EigenSystem2D(const QPW_Cell& cell, const QPW_DeformVectors&
73 defvecs)
74 {
75 const auto& decomp = cell.m_Decomposition;
76 const auto& invar = cell.m_Invariables;
77
78 pairs[0].l = 1 + 3 / std::pow(decomp.sigma(0), 4.0);
79 pairs[0].e = defvecs.d1;
80 pairs[1].l = 1 + 3 / std::pow(decomp.sigma(1), 4.0);
81 pairs[1].e = defvecs.d2;
82 pairs[2].l = 1 + 1 / std::pow(invar.I3, 2.0) + invar.I2 / std::pow(invar.I3, 3.0);
83 pairs[2].e = defvecs.l;
84 pairs[3].l = 1 + 1 / std::pow(invar.I3, 2.0) - invar.I2 / std::pow(invar.I3, 3.0);
85 pairs[3].e = defvecs.t;
86 }
87
88 namespace eigensys
89 {
90 QPW_Cell::QPW_Cell(const Eigen::Matrix2d& dm) : Dm(dm)
91 {
92 // DmINV = [[u, w]T [v, t]T], pfq_pxq has nothing to do with Ds
93 Eigen::Matrix2d DmINV = Dm.inverse();
94 double u = DmINV(0, 0);
95 double v = DmINV(0, 1);
96 double w = DmINV(1, 0);
97 double t = DmINV(1, 1);
98 double s1 = u + w;
99 double s2 = v + t;
100
101 // pfq pfq pfq pfq pfq pfq
102 // --- --- --- --- --- ---
103 // px1x px1y px2x px2y px3x px3y
104 pfq_pxq << -s1, 0.0, u, 0.0, w, 0.0,
105 0.0, -s1, 0.0, u, 0.0, w,
106 -s2, 0.0, v, 0.0, t, 0.0,
107 0.0, -s2, 0.0, v, 0.0, t;
108 }
109
110 std::pair<QPW_Cell::vec6d, QPW_Cell::mat6d> QPW_Cell::CalculateGradNHess(const Eigen
111 ::Matrix2d& Ds)
112 {

```

```

112 // volume weight
113 double volumeWeight = GetVolumeWeight();
114
115 // 1. Prepare F
116 Eigen::Matrix2d F = Ds * Dm.inverse();
117
118 // 2. Variables (decomposition, invariables, vectors)
119 auto defvecs = CalculateVars(F);
120
121 // 3. energy-deform gradient
122 auto pPSIq_pfq = CalculateEnergyDeformGrad(m_Invariables, defvecs);
123 vec6d GRAD = volumeWeight * pfq_pxq.transpose() * pPSIq_pfq;
124
125 // 4. energy-deform hessian
126 QPW_EigenSystem2D eigensys(*this, defvecs);
127 auto p2PSIq_pfq2 = CalculateEnergyDeformHess(eigensys);
128 mat6d HESS = volumeWeight * pfq_pxq.transpose() * p2PSIq_pfq2 * pfq_pxq;
129
130 return std::make_pair(GRAD, HESS);
131 }
132
133 double QPW_Cell::CalculateEnergy(const Eigen::Matrix2d& Ds)
134 {
135 // 1. Prepare F
136 Eigen::Matrix2d F = Ds * Dm.inverse();
137
138 // 2. Variables (decomposition, invariables)
139 CalculateVars(F);
140
141 // 3. energy
142 return GetVolumeWeight() * CalculateEnergy(m_Invariables);
143 }
144
145 double QPW_Cell::CalculateEnergy(const QPW_Invariables& invars) const
146 {
147 return (invars.I2 + invars.I2 / std::pow(invars.I3, 2.0)) / 2.0;
148 }
149
150 QPW_DeformVectors QPW_Cell::CalculateVars(const Eigen::Matrix2d& F)
151 {
152 m_Decomposition.CalculateSVDnPolar(F);
153 m_Invariables.CalculateInvariants(m_Decomposition.S);
154 return QPW_DeformVectors(F, m_Decomposition);
155 }
156
157 Eigen::Vector4d QPW_Cell::CalculateEnergyDeformGrad(const QPW_Invariables& invars,
158 const QPW_DeformVectors& defvecs) const
159 {
160 Eigen::Vector4d pPSIq_pfq = Eigen::Vector4d::Zero();

```

```

160 const auto& invar = m_Invariables;
161
162 double pPSI_pIs[3] = { 0.0, // pPSI / pI1 = 0
163 (1.0 + 1.0 / std::pow(invar.I3, 2.0)) / 2.0, // pPSI / pI2 = (1 + 1 / I3^2) / 2
164 - invar.I2 / std::pow(invar.I3, 3.0) }; // pPSI / pI3 = -I2 / I3^3
165
166 Eigen::Vector4d pI_pFqs[3] = { defvecs.r, // pI1 / pfq = r
167 defvecs.f * 2.0, // pI2 / pfq = 2f
168 defvecs.g }; // pI3 / pfq = g
169
170 return std::inner_product(pPSI_pIs, pPSI_pIs + 3, pI_pFqs, Eigen::Vector4d{ Eigen::Vector4d::Zero() });
171 }
172
173 Eigen::Matrix4d QPW_Cell::CalculateEnergyDeformHess(const QPW_EigenSystem2D& eigensys)
174) const
175 {
176 return std::accumulate(eigensys.pairs.begin(), eigensys.pairs.end(),
177 Eigen::Matrix4d{ Eigen::Matrix4d::Zero() },
178 [](Eigen::Matrix4d sum, const QPW_EigenValVecPair& pair)
179 { return sum + std::max(pair.l, 0.0) * (pair.e * pair.e.transpose()); });
180 }
```

Listing A.6: Util\_EnergyEigensystems.h

```

1 // ===== Util_EnergyEigensystems.h =====
2 #pragma once
3
4 #include <tuple>
5 #include <vector>
6 #include <filesystem>
7 #include <Eigen/Dense>
8 #include <Eigen/Sparse>
9
10 #include <Eigen/SparseLU>
11
12 #include " ../PolyMesh/include/PolyMesh/PolyMesh.h"
13
14 #include "Util_DataStructure.h"
15
16 // 2D, Symmetric Dirichlet Energy Only
17 namespace eigensys
18 {
19 class ProjectNewtonSolver
20 {
21 public:
22 ProjectNewtonSolver();
23 void PresetMeshUV(acamcad::polymesh::PolyMesh* mesh, const Eigen::SparseMatrix<
```

```

1 double>& UVmat);
2 bool UpdateMeshUV(acamcad::polymesh::PolyMesh* mesh);
3 void SaveEnergies(const std::filesystem::path& filePath) const;
4
5
6
7 private:
8 void ConstrainUV(acamcad::polymesh::PolyMesh* mesh, const Eigen::VectorXd& UVs);
9
10
11 std::pair<double, Eigen::VectorXd> Line_Search(
12 acamcad::polymesh::PolyMesh* mesh,
13 const Eigen::VectorXd& d,
14 const Eigen::VectorXd& grad,
15 double lastEnergy,
16 double gamma, double c, double a0);
17
18
19 // procedure function
20 double CalculateEnergySD_2D(acamcad::polymesh::PolyMesh* mesh, const Eigen::VectorXd& UVs) const;
21 std::tuple<Eigen::VectorXd, Eigen::SparseMatrix<double>> CalculateEnergyDerivative(
22 acamcad::polymesh::PolyMesh* mesh, const Eigen::VectorXd& UVs);
23
24 double CalculateNoFlipoverStep(acamcad::polymesh::PolyMesh* mesh, const Eigen::VectorXd& grad) const;
25
26
27 private:
28 // data meber
29 std::vector<QPW_Cell> m_CellList;
30 Eigen::VectorXd m_UVList;
31
32 // statistics
33 size_t m_Iters{ 0 };
34 double m_Energy{ 0.0 };
35
36 std::vector<double> m_EnergyRecord;
37
38 // utility
39 Eigen::SimplicialLDLT<Eigen::SparseMatrix<double>> m_LDLTSolver;
40
41 // control
42 bool m_FirstUpdate{ true };
43 };
44 }

```

Listing A.7: Util\_EnergyEigensystems.cpp

```

1 // ===== Util_EnergyEigensystems.cpp =====
2 #include "Util_EnergyEigensystems.h"
3
4 #include <numeric>
5 #include <algorithm>

```

```

6 #include <fstream>
7
8 namespace eigensys
9 {
10 // constrain UV in [0, 1]x[0, 1]
11 void ProjectNewtonSolver::ConstrainUV(acamcad::polymesh::PolyMesh* mesh, const Eigen
12 ::VectorXd& UVs)
13 {
14 double xmin = std::numeric_limits<double>::max();
15 double xmax = -std::numeric_limits<double>::max();
16 double ymin = std::numeric_limits<double>::max();
17 double ymax = -std::numeric_limits<double>::max();
18
19 size_t numV = mesh->vertices().size();
20 for (size_t vertID = 0; vertID < numV; ++vertID)
21 {
22 double x = UVs(vertID * 2 + 0);
23 double y = UVs(vertID * 2 + 1);
24
25 xmin = std::min(xmin, x);
26 xmax = std::max(xmax, x);
27 ymin = std::min(ymin, y);
28 ymax = std::max(ymax, y);
29 }
30
31 for (size_t vertID = 0; vertID < numV; ++vertID)
32 {
33 auto* vert = mesh->vert(vertID);
34 float UVx = static_cast<float>((UVs(2 * vertID + 0) - xmin) / (xmax - xmin));
35 float UVy = static_cast<float>((UVs(2 * vertID + 1) - ymin) / (ymax - ymin));
36 vert->setTexture(UVx, UVy, 0.0);
37 vert->setPosition(UVx, UVy, 0.0);
38 }
39 }
40
41 ProjectNewtonSolver::ProjectNewtonSolver()
42 {
43 m_UVList.setZero();
44 m_EnergyRecord.reserve(1000);
45 m_LDLTSolver.setShift(std::numeric_limits<float>::epsilon()); // necessary
46 }
47
48 // x comes from tutte's embedding results
49 void ProjectNewtonSolver::PresetMeshUV(acamcad::polymesh::PolyMesh* mesh, const Eigen
50 ::SparseMatrix<double>& UVmat)
51 {
52 assert(mesh != nullptr);
53 m_Iters = 0;
54 m_FirstUpdate = true;

```

```

53
54 size_t numF = mesh->polyfaces().size();
55 size_t numV = mesh->vertices().size();
56
57 m_UVList = Eigen::VectorXd(2 * numV);
58 m_UVList.setZero();
59 for (size_t vertID = 0; vertID < numV; ++vertID)
60 {
61 m_UVList(2 * vertID + 0) = UVmat.coeff(vertID, 0);
62 m_UVList(2 * vertID + 1) = UVmat.coeff(vertID, 1);
63 }
64
65 m_CellList.clear();
66 m_CellList.reserve(numF);
67 for (size_t faceID = 0; faceID < numF; ++faceID)
68 {
69 auto faceVerts = mesh->polygonVertices(mesh->polyface(faceID));
70 auto* v0 = faceVerts[0];
71 auto* v1 = faceVerts[1];
72 auto* v2 = faceVerts[2];
73
74 auto v10 = v1->position() - v0->position();
75 auto v20 = v2->position() - v0->position();
76 // y
77 // ^
78 // | v2
79 // .<---.
80 // ^ /|\ \
81 // | / | \
82 // |/ \ \
83 // |/ V \
84 // .==>.--->.-> x
85 // v0 vp v1
86 Eigen::Vector2d P0{ 0, 0 };
87 Eigen::Vector2d P1{ v10.norm(), 0 };
88 Eigen::Vector2d P2{ v10.dot(v20) / v10.norm(), v10.cross(v20).norm() / v10.norm() };
89
90 Eigen::Matrix2d Dm;
91 Dm << P1 - P0, P2 - P0;
92 m_CellList.emplace_back(Dm);
93 }
94
95 m_EnergyRecord.clear();
96 }
97
98 // Please make sure that the UV point are stored in the texture coordinate
99 bool ProjectNewtonSolver::UpdateMeshUV(acamcad::polymesh::PolyMesh* mesh)
100 {

```

```

101 if (mesh == nullptr)
102 {
103 std::cout << "the mesh pointer is NULL!\n";
104 return false;
105 }
106
107 size_t numV = mesh->vertices().size();
108 size_t numF = mesh->polyfaces().size();
109
110 std::cout << "[ProjectNewton] ";
111
112 auto [G, H] = CalculateEnergyDerivative(mesh, m_UVList);
113
114 std::cout << "max |Gi| = " << G.cwiseAbs().maxCoeff() << "\t";
115
116 if (G.cwiseAbs().maxCoeff() < 1.0e-4)
117 {
118 std::cout << "=====CONVERGE!!!===== \n";
119 m_Iters = 0;
120 return false;
121 }
122
123 if (m_FirstUpdate)
124 {
125 m_Energy = CalculateEnergySD_2D(mesh, m_UVList);
126 m_EnergyRecord.emplace_back(m_Energy);
127 m_FirstUpdate = false;
128 }
129
130 m_LDLTSolver.compute(H);
131 Eigen::VectorXd d = m_LDLTSolver.solve(-G);
132
133 double step = CalculateNoFlipoverStep(mesh, d);
134 std::cout << "A = " << step << "\t";
135
136 auto [energy, updatedUV] = Line_Search(mesh, d, G, m_Energy, 0.8, 1.0e-4, std::min(
137 step * 0.99, 1.0));
138 m_EnergyRecord.emplace_back(energy);
139
140 m_Energy = energy;
141 m_UVList = updatedUV;
142
143 ConstrainUV(mesh, updatedUV);
144
145 return true;
146}
147 void ProjectNewtonSolver::SaveEnergies(const std::filesystem::path& filePath) const
148 {

```

```

149 std::ofstream file(filePath);
150 if (!file.is_open()) {
151 file.close();
152 assert(false, "file open error!");
153 }
154
155 for (double energy : m_EnergyRecord)
156 file << energy << ", ";
157
158 file.close();
159 }
160
161 std::pair<double, Eigen::VectorXd> ProjectNewtonSolver::Line_Search(
162 acamcad::polymesh::PolyMesh* mesh,
163 const Eigen::VectorXd& d,
164 const Eigen::VectorXd& grad,
165 double lastEnergy,
166 double gamma, double c, double a0)
167 {
168 double alpha = a0;
169 double currentEnergy = lastEnergy;
170
171 const int MAX_LOOP_SIZE = 64;
172
173 auto updatedUVs = m_UVList;
174 size_t iter = 0;
175 for (iter = 0; iter < MAX_LOOP_SIZE; ++iter)
176 {
177 updatedUVs = m_UVList + alpha * d;
178 currentEnergy = CalculateEnergySD_2D(mesh, updatedUVs);
179
180 if (currentEnergy <= lastEnergy + c * alpha * d.dot(grad)) break;
181 alpha *= gamma;
182 }
183 std::cout << "Iter = " << ++m_Iters << "\tEo = " << lastEnergy
184 << "\tEn = " << currentEnergy << "\talpha = " << alpha << "\n";
185
186 return std::make_pair(currentEnergy, updatedUVs);
187 }
188
189 double ProjectNewtonSolver::CalculateEnergySD_2D(
190 acamcad::polymesh::PolyMesh* mesh,
191 const Eigen::VectorXd& UVs) const
192 {
193 double energy = 0.0;
194 size_t numF = mesh->polyfaces().size();
195
196 for (size_t faceID = 0; faceID < numF; ++faceID)
197 {

```

```

198 auto ids = PolyFaceVertIdxs(mesh, faceID) * 2;
199
200 Eigen::Matrix2d Ds = Eigen::Matrix2d::Zero();
201 Ds << UVs.segment(ids(1), 2) - UVs.segment(ids(0), 2),
202 UVs.segment(ids(2), 2) - UVs.segment(ids(0), 2);
203
204 auto& cell = m_CellList[faceID];
205 energy += cell.GetVolumeWeight() * cell.GetLastEnergy();
206 }
207
208 return energy;
209 }
210
211 std::tuple<Eigen::VectorXd, Eigen::SparseMatrix<double>>
212 ProjectNewtonSolver::CalculateEnergyDerivative(
213 acamcad::polymesh::PolyMesh* mesh, const Eigen::VectorXd& UVs)
214 {
215 size_t numV = mesh->vertices().size();
216 size_t numF = mesh->polyfaces().size();
217
218 Eigen::VectorXd GRAD(2 * numV);
219 GRAD.setZero();
220
221 Eigen::SparseMatrix<double> HESS(2 * numV, 2 * numV);
222 HESS.setZero();
223 std::vector<Eigen::Triplet<double>> HESSStrips;
224 HESSStrips.reserve(6 * numF);
225
226 for (size_t faceID = 0; faceID < numF; ++faceID)
227 {
228 auto ids = PolyFaceVertIdxs(mesh, faceID) * 2;
229 int GlobalCoord[6] = {ids(0), ids(0) + 1, ids(1), ids(1) + 1, ids(2), ids(2) + 1};
230
231 Eigen::Matrix2d Ds;
232 Ds << UVs.segment(ids(1), 2) - UVs.segment(ids(0), 2),
233 UVs.segment(ids(2), 2) - UVs.segment(ids(0), 2);
234
235 auto [gradq, hessq] = m_CellList[faceID].CalculateGradNHess(Ds);
236
237 for (int n = 0; n < 6; ++n)
238 {
239 GRAD(GlobalCoord[n]) += gradq(n);
240 for (int m = 0; m < 6; ++m)
241 HESSStrips.emplace_back(GlobalCoord[n], GlobalCoord[m], hessq(n, m));
242 }
243 assert(!std::isnan(GRAD.norm()));
244 }
245

```

```

246 HESS.setFromTriplets(HESStrips.begin(), HESStrips.end());
247 HESS.makeCompressed();
248 return std::make_tuple(GRAD, HESS);
249 }
250
251 double ProjectNewtonSolver::CalculateNoFlipoverStep(acamcad::polymesh::PolyMesh* mesh
252 , const Eigen::VectorXd& grad) const
253 {
254 size_t numF = mesh->polyfaces().size();
255
256 const auto& UVs = m_UVList;
257 double minStep = 1.0;
258
259 for (size_t faceID = 0; faceID < numF; ++faceID)
260 {
261 auto ids = PolyFaceVertIdxs(mesh, faceID) * 2;
262 Eigen::Matrix2d Ds = Eigen::Matrix2d::Zero();
263 Ds << UVs.segment(ids(1), 2) - UVs.segment(ids(0), 2),
264 UVs.segment(ids(2), 2) - UVs.segment(ids(0), 2);
265
266 Eigen::Matrix2d D = Eigen::Matrix2d::Zero();
267 D << grad.segment(ids(1), 2) - grad.segment(ids(0), 2),
268 grad.segment(ids(2), 2) - grad.segment(ids(0), 2);
269
270 double A = D.determinant();
271 double B = D.determinant() + Ds.determinant() - (Ds - D).determinant();
272 double C = Ds.determinant();
273
274 double t1 = 0.0;
275 double t2 = 0.0;
276 if (std::abs(A) > 1.0e-10)
277 {
278 double Delta = B * B - 4 * A * C;
279 if (Delta <= 0) continue;
280
281 double delta = std::sqrt(Delta); // delta ≥ 0
282 if (B ≥ 0)
283 {
284 double bd = -B - delta;
285 t1 = 2 * C / bd;
286 t2 = bd / (2 * A);
287 }
288 else
289 {
290 double bd = -B + delta;
291 t1 = bd / (2 * A);
292 t2 = (2 * C) / bd;
293 }
294 }
295 }
296 }

```

```
294 if (A < 0) std::swap(t1, t2); // make t1 > t2
295 minStep = (t1 > 0) ? (std::min(minStep, t2 > 0 ? t2 : t1)) : minStep;
296 }
297 else
298 {
299 t1 = -C / B;
300 // avoid divide-by-zero
301 minStep = (B == 0) ? minStep : ((t1 > 0) ? t1 : minStep);
302 }
303 }
304 return minStep;
305 }
306 }
```

## 附录 B 致谢

- 在此感谢古明地绿（QQ 746530916）同学提出的基于无翻转策略初始化线搜索步长参数  $\alpha_{\text{upper}}^k$  的方案。
- 在此感谢 zeno（QQ 117127143）同学帮助推导了方程 1.23 的另一种形式。即如果矩阵  $\mathbf{D}^k$  可逆，令  $\mathbf{A}_q = \mathbf{D}_s^k [\mathbf{D}^k]^{-1}$ ，则有关于  $\alpha_q^k$  的一元二次方程

$$(\alpha_q^k)^2 + \text{tr}(\mathbf{A}_q)\alpha_q^k + \det(\mathbf{A}_q) = 0$$

成立。在二维情况下上式和公式 1.23 等价，而且此形式可以推广至三维情形<sup>1</sup>。但是考虑到浮点精度不足和  $\mathbf{D}^k$  不可逆的潜在可能，本文仍旧采用公式 1.23 的形式求解。

---

<sup>1</sup>存疑，本文没有证明这一点。如果有哪位大佬能看到这份报告，请帮忙证明一下三维情形有没有和公式 1.23 类似的表达，感激不尽！

## Bibliography

- [1] 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>.
- [2] Theodore Kim and David Eberle. “Dynamic Deformables: Implementation and Production Practicalities (Now with Code!)” In: *ACM SIGGRAPH 2022 Courses*. SIGGRAPH ’22. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2022. ISBN: 9781450393621. DOI: [10.1145/3532720.3535628](https://doi.org/10.1145/3532720.3535628). URL: <https://doi.org/10.1145/3532720.3535628>.
- [3] William Thomas Tutte. “Convex representations of graphs”. In: *Proceedings of the London Mathematical Society* 3.1 (1960), pp. 304–320.
- [4] William Thomas Tutte. “How to draw a graph”. In: *Proceedings of the London Mathematical Society* 3.1 (1963), pp. 743–767.
- [5] Michael S Floater. “Parametrization and smooth approximation of surface triangulations”. In: *Computer aided geometric design* 14.3 (1997), pp. 231–250.
- [6] 刘浩洋 et al. 最优化：建模、算法与理论. 高等教育出版社, 2020.
- [7] Wikipedia. *Finite strain theory*. [https://en.wikipedia.org/wiki/Finite\\_strain\\_theory#Deformation\\_gradient\\_tensor](https://en.wikipedia.org/wiki/Finite_strain_theory#Deformation_gradient_tensor). 2022.
- [8] Isaac Chao et al. “A Simple Geometric Model for Elastic Deformations”. In: *ACM Trans. Graph.* 29.4 (July 2010). ISSN: 0730-0301. DOI: [10.1145/1778765.1778775](https://doi.org/10.1145/1778765.1778775). URL: <https://doi.org/10.1145/1778765.1778775>.
- [9] Jason Smith and Scott Schaefer. “Bijective Parameterization with Free Boundaries”. In: *ACM Trans. Graph.* 34.4 (July 2015). ISSN: 0730-0301. DOI: [10.1145/2766947](https://doi.org/10.1145/2766947). URL: <https://doi.org/10.1145/2766947>.
- [10] Kai Hormann and Günther Greiner. *MIPS: An efficient global parametrization method*. Tech. rep. Erlangen-Nuernberg Univ (Germany) Computer Graphics Group, 2000.
- [11] Michael Rabinovich et al. “Scalable Locally Injective Mappings”. In: *ACM Trans. Graph.* 36.2 (Apr. 2017). ISSN: 0730-0301. DOI: [10.1145/2983621](https://doi.org/10.1145/2983621). URL: <https://doi.org/10.1145/2983621>.