

HW2

原理

核心原理

本次作业的目的是实现文章[Analytic Eigensystems for Isotropic Distortion Energies](#)中的算法。

该文章的基本算法框架为projected newton法，伪代码如下所示：

ALGORITHM 1: Projected Newton Pseudocode. Our approach allows `Eval_Energy_EigenSystem(U, Σ, V)` to be implemented in closed-form.

```
Function Projected_Newton_Solver(x₀)
    for i ← 0 to n do
        bᵢ ← ∇Ψ(xᵢ) // Equation (5a)
        if ||bᵢ||∞ ≤ 10⁻⁴ then
            return xᵢ
        end
        Hᵢ ← Project_Hessian(xᵢ)
        dᵢ ← -Hᵢ⁻¹bᵢ
        αᵢ ← Line_Search(xᵢ, dᵢ)
        xᵢ₊₁ ← xᵢ + αᵢdᵢ
    end
    return xₙ₊₁

Function Project_Hessian(x)
    H ← 0
    for every quadrature point q do
        F ← Compute_Deformation_Gradient(q, x)
        f ← vec(F)
        {U, Σ, V} ← Compute_SVD(F)
        {λᵢ, eᵢ} ← Eval_Energy_EigenSystem(U, Σ, V)
        Hᵢ ← ∑ᵢ max(λᵢ, 0) eᵢ eᵢᵀ
        H ← H + |q| (df / dx)ᵀ Hᵢ (df / dx) // Equation (5b)
    end
    return H
```

该文章的核心内容为如何解析的构造半正定矩阵 H^+ 。

在几何优化中使用的能量函数可以通过形变梯度张量 F 来表达。

在该文章中，提出使用 F 的极分解 $F = RS$ 中出现的不变量来解析的构造能量函数的Hessian矩阵。其中 R 代表了刚性旋转， S 则是对应的右拉伸张量。使用的三个不变量如下：

$$\begin{aligned} I_1 &= \text{tr}(S) = \sum_i \sigma_i \\ I_2 &= \|S\|^2 = \sum_i \sigma_i^2 \\ I_3 &= \det(S) = \prod_i \sigma_i \end{aligned}$$

通过这三个不变量，我们便可以构造出任意能量函数的梯度以及Hessian矩阵。

例如对于对称Dirichlet能量，我们有 $\Psi^{2D} = (\|F\|^2 + \|F^{-1}\|^2) = I_2 + I_2/I_3^2$ ，

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

从而对应的Hessian矩阵为 $\sum_i \max(\lambda_i, 0) e_i e_i^\top$ 。

最后，再由

$$\begin{cases} \frac{\partial \Psi}{\partial \mathbf{x}} = \sum_q |q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \frac{\partial \Psi_q}{\partial \mathbf{f}_q}, \\ \frac{\partial^2 \Psi}{\partial \mathbf{x}^2} = \sum_q |q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \left(\frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} \right) \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}}. \end{cases}$$

即可求出完整的梯度以及Hessian矩阵。

因此，我们剩下的任务便是计算形变梯度张量 F 以及对应的 $\frac{\partial F}{\partial \mathbf{x}}$ 。

F与 $\frac{\partial F}{\partial \mathbf{x}}$ 的计算

F 描述了从点 \bar{x} 到点 x 的旋转和缩放，为了计算 F ，我们将原始三角形和形变后的三角形中的一个顶点移至原点，此时，我们有：

$$\begin{aligned}\bar{x}_0 &\rightarrow \bar{o}_0 = [0 \quad 0]^\top \\ \bar{x}_1 &\rightarrow \bar{o}_1 = \bar{x}_1 - \bar{x}_0 \\ \bar{x}_2 &\rightarrow \bar{o}_2 = \bar{x}_2 - \bar{x}_0 \\ x_0 &\rightarrow o_0 = [0 \quad 0]^\top \\ x_1 &\rightarrow o_1 = x_1 - x_0 \\ x_2 &\rightarrow o_2 = x_2 - x_0\end{aligned}$$

F 将 \bar{o}_i 映射到 o_i ，即满足以下三个方程：

$$F\bar{o}_0 = o_0 \quad F\bar{o}_1 = o_1 \quad F\bar{o}_2 = o_2.$$

其中，第一个方程对任意的 F 均成立，因此，我们只需求解由剩余两个方程组成的方程组即可，即：

$$\begin{aligned}F \begin{bmatrix} \bar{x}_1 - \bar{x}_0 & | & \bar{x}_2 - \bar{x}_0 \end{bmatrix} &= \begin{bmatrix} x_1 - x_0 & | & x_2 - x_0 \end{bmatrix} \\ F\mathbf{D}_m &= \mathbf{D}_s\end{aligned}$$

从而，我们有 $F = \mathbf{D}_s \mathbf{D}_m^{-1}$ 。注意其中的 \mathbf{D}_m 只依赖于mesh的结构，即在我们的参数化过程中并不会改变，因此，我们可以一开始先将其逆计算并存储，这样在计算 F 时，不需要再次计算逆。

根据上面公式，我们有

$$\frac{\partial F}{\partial \mathbf{x}} = \frac{\partial \mathbf{D}_s}{\partial \mathbf{x}} \mathbf{D}_m^{-1}$$

参考在[这个课程](#)中提到的计算方式，我们可以写出 $\frac{\partial \mathbf{D}_s}{\partial \mathbf{x}}$ 的各分量如下：

$$\begin{aligned}\frac{\partial \mathbf{D}_s}{\partial x_0} &= \begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix} & \frac{\partial \mathbf{D}_s}{\partial x_1} &= \begin{bmatrix} 0 & 0 \\ -1 & -1 \end{bmatrix} \\ \frac{\partial \mathbf{D}_s}{\partial x_2} &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & \frac{\partial \mathbf{D}_s}{\partial x_3} &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \\ \frac{\partial \mathbf{D}_s}{\partial x_4} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} & \frac{\partial \mathbf{D}_s}{\partial x_5} &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\end{aligned}$$

即最终的 $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ 是一个 4×6 的矩阵，其中每一项元素均只与 \mathbf{D}_m^{-1} 相关，因此，我们也可以类似的将其预先计算出来，这样在运行时便不需要再次计算。

Line search算法

在确定下降方向后，我们便需要进一步估算恰当的步长，使得目标函数下降得足够多，我们采用了 Wolfe准则来衡量目标函数下降程度。算法伪代码如下：

```
Input: initial step size  $\bar{\alpha} > 0, \rho, c \in (0, 1)$ 
do
|    $\alpha \leftarrow \rho\alpha$  ;
while  $f(\mathbf{x}_k + \alpha \mathbf{p}_k) \leq f(\mathbf{x}_k) + c\alpha \nabla f_k^\top \mathbf{p}_k$ ;
```

为了保证最终的无翻转映射，我们需要对初始步长进行限制，对于一个三角形，当其发生翻转时，我们有

$$\det \begin{bmatrix} (\mathbf{u}_2 + \mathbf{v}_2\alpha) - (\mathbf{u}_1 + \mathbf{v}_1\alpha) \\ (\mathbf{u}_3 + \mathbf{v}_2\alpha) - (\mathbf{u}_1 + \mathbf{v}_1\alpha) \end{bmatrix} = 0$$

因此，我们只需求解该方程，将其最小的正根作为最大步长即可保证无翻转。

代码实现

这次作业分为了较多文件，在 `eigen_system.h` 中定义了本作业实现文章中的核心特征值系统，其中存储了四个特征值以及对应的特征向量，以及相应的不变量。

在 `line_search.h` 文件中则是实现了基本的线搜索算法和保证无翻转的线搜索算法。

在 `parametrization.h` 中则是实现了算法的核心部分。首先是 `projected_newton_precompute` 函数，在该函数中我们将在原理部分提及的可预先计算的 \mathbf{D}_m^{-1} 、 $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ 以及各三角形的面积进行计算并保存，这样可以减少后续所需的运算。

整个算法的主体在 `projected_newton_solver` 函数中，在该函数中，我们首先计算出梯度向量以及 Hessian 矩阵。然后求解出线搜索的搜索方向，最后通过线搜索算法找出新的 uv 值。其中梯度向量以及 Hessian 矩阵的计算是在 `project_gradient_and_hessian` 函数中完成的。该函数按以下公式

$$\begin{cases} \frac{\partial \Psi}{\partial \mathbf{x}} = \sum_q |q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \frac{\partial \Psi_q}{\partial \mathbf{f}_q}, \\ \frac{\partial^2 \Psi}{\partial \mathbf{x}^2} = \sum_q |q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \left(\frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} \right) \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}} \end{cases}$$

计算相应的梯度向量以及 Hessian 矩阵。对于其中每个三角形具体的 $|q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \frac{\partial \Psi_q}{\partial \mathbf{f}_q}$ 和 $|q| \frac{\partial \mathbf{f}_q^\top}{\partial \mathbf{x}} \left(\frac{\partial^2 \Psi_q}{\partial \mathbf{f}_q^2} \right) \frac{\partial \mathbf{f}_q}{\partial \mathbf{x}}$ 是在 `deformation_gradient_and_hessian` 函数中计算的，然后在 `project_gradient_and_hessian` 函数中进行累计求和。为了加快速度，我们使用了 `openmp` 进行并行化处理。注意到由于搜索步长较小，实际上对于每个三角形其梯度和 Hessian 矩阵可能数值上相差较大量级，同时由于相当一部分分量较小，最后相加时可能会产生误差从而导致最终算法数值上的不稳定。因此，在梯度向量的计算时，我们采用 [Kahan summation 算法](#) 求和来消除误差；但在构建 Hessian 矩阵时，我们采用的是 `Eigen` 库中的稀疏矩阵以及对应的 `setFromTriplets` 函数来计算的，其舍入的数值精度为 10^{-12} ，仍然会产生误差。

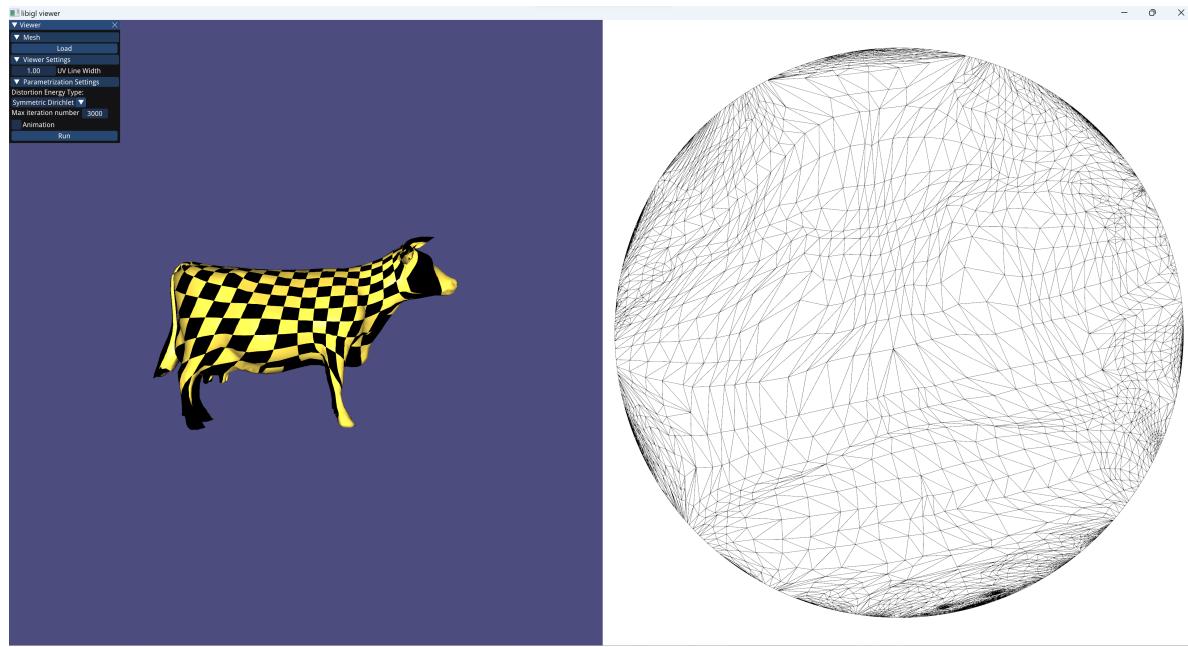
在 `util.h` 文件中则是实现了之前提及的各个函数需要使用的一些辅助性函数，如实现 Kahan summation 算法的 `kahan_sum` 函数以及用于计算线搜索中最大步长的 `compute_max_step_from_singularities` 函数等。

在 `symmetric_dirichlet.*`、`arap.*` 以及 `mips.*` 中则实现了对应能量的具体特征值、能量以及梯度计算。

最后在 `main.cpp` 中则是实现了整个算法的GUI界面。

结果

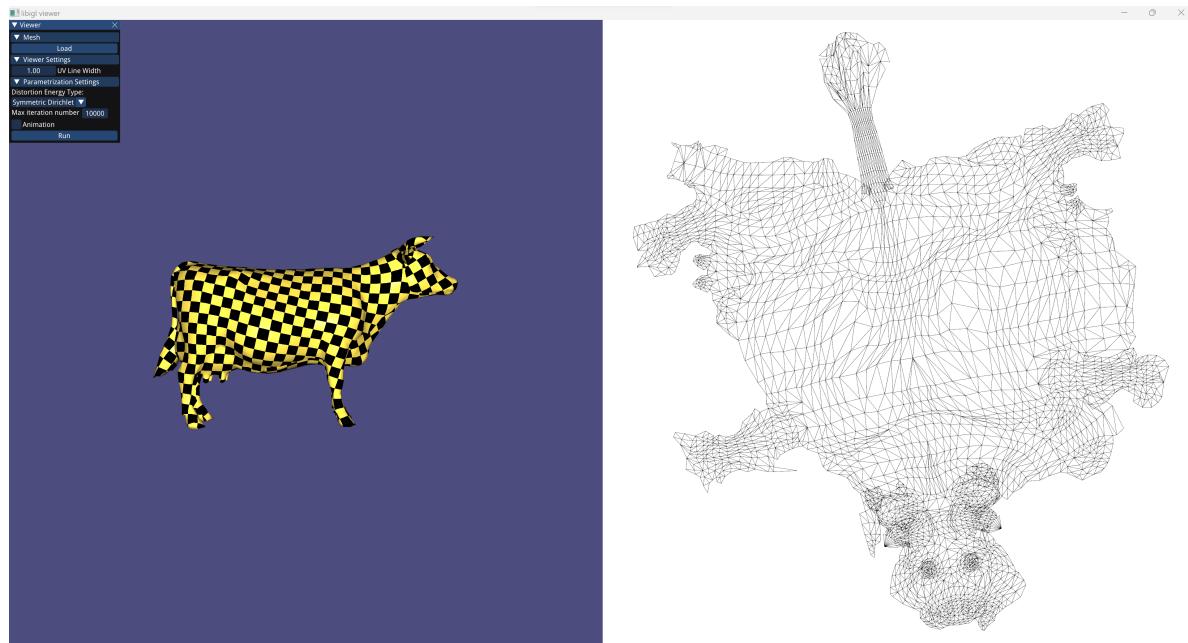
编译运行后出现如下图所示主界面

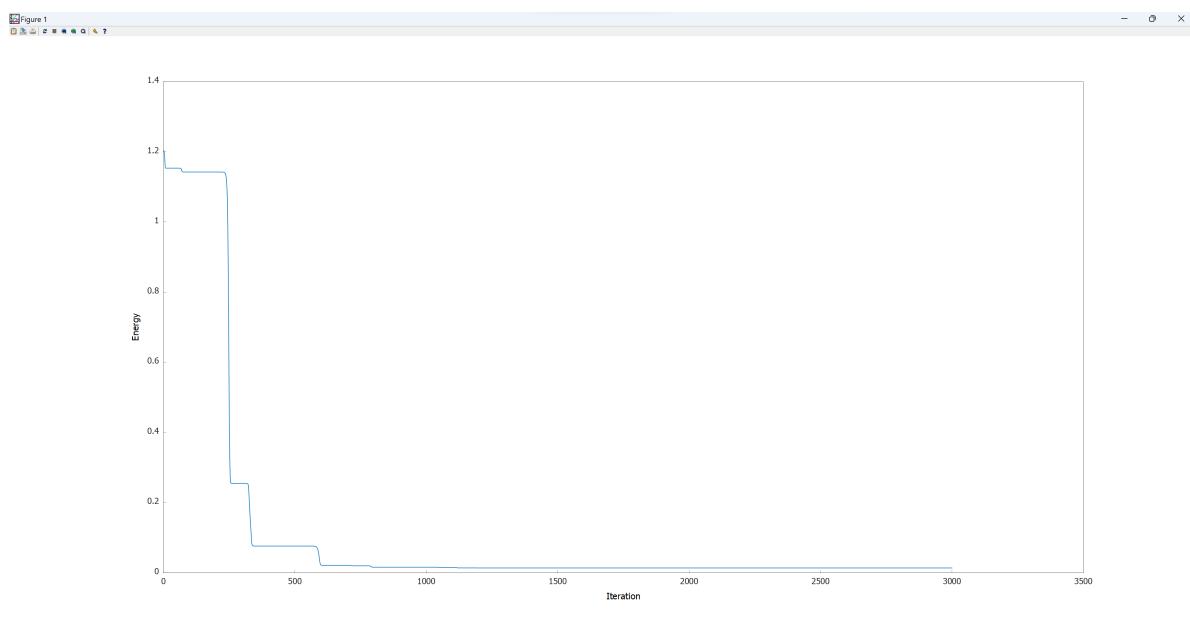
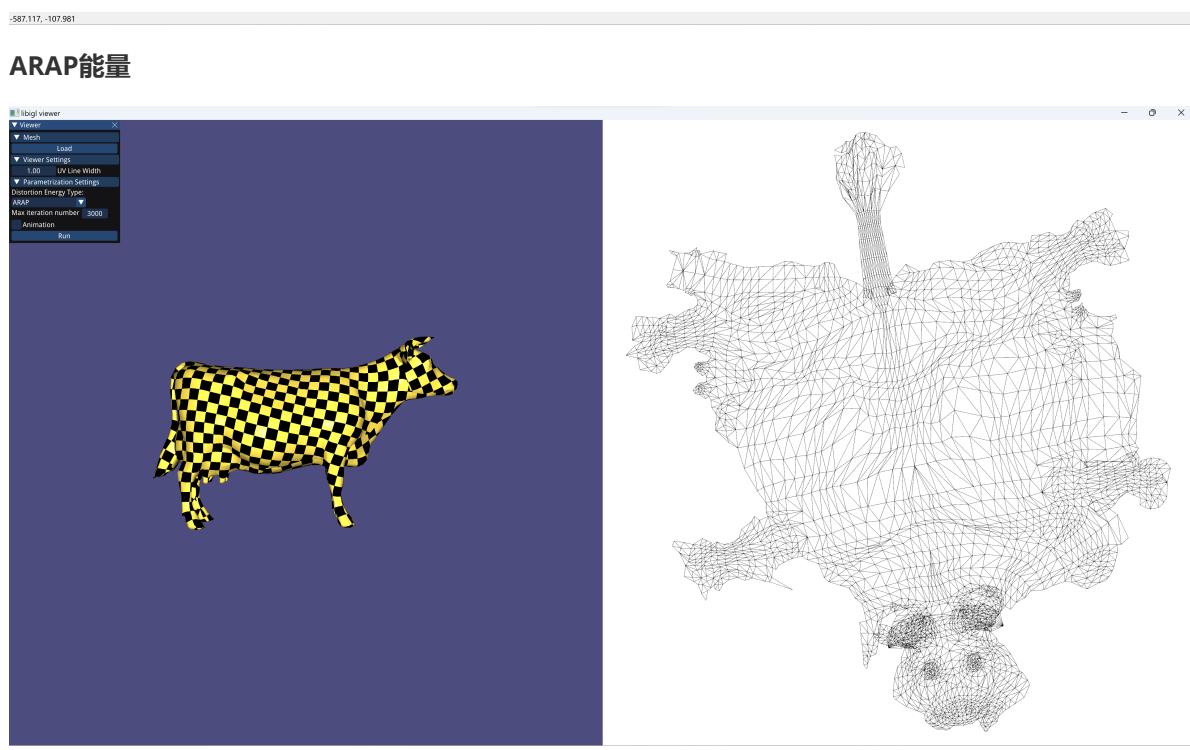
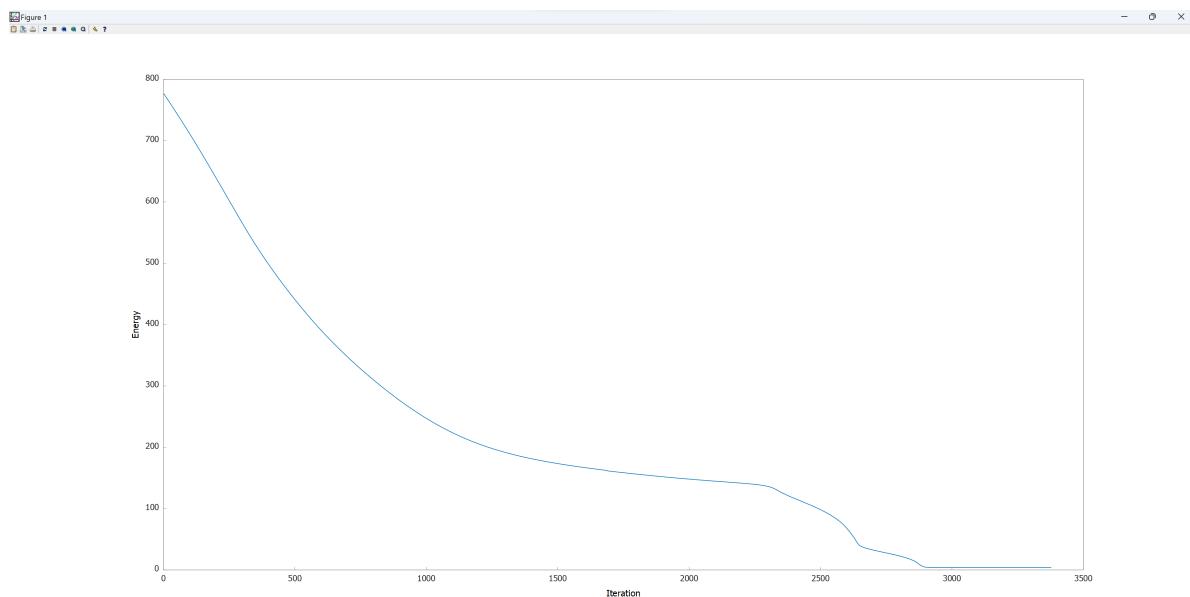


通过 `Load` 按钮加载网格体文件后会显示 Tutte 参数化的结果，在设置好能量类型以及最大迭代次数后，点击运行按钮即可执行参数化过程。目前支持ARAP、对称Dirichlet以及MISP三种能量。此外，如果环境中存在 `gnuplot`，则在参数化结束或，会生成能量随迭代次数变化的图。最后，勾选 `Animation` 选项后运行则会以动画的形式展示参数化过程。以下为参数化结果：

Cow

对称Dirichlet能量

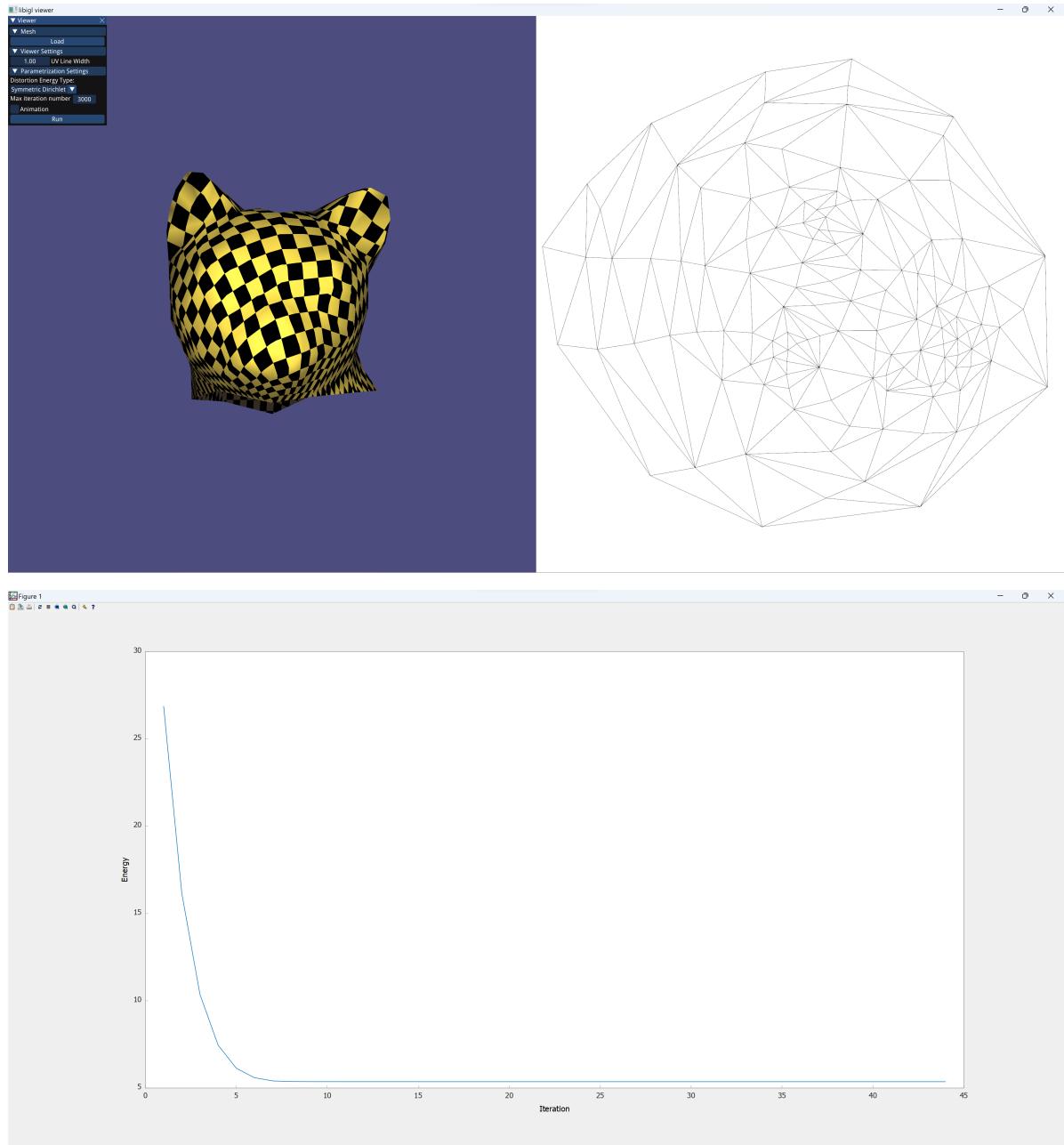




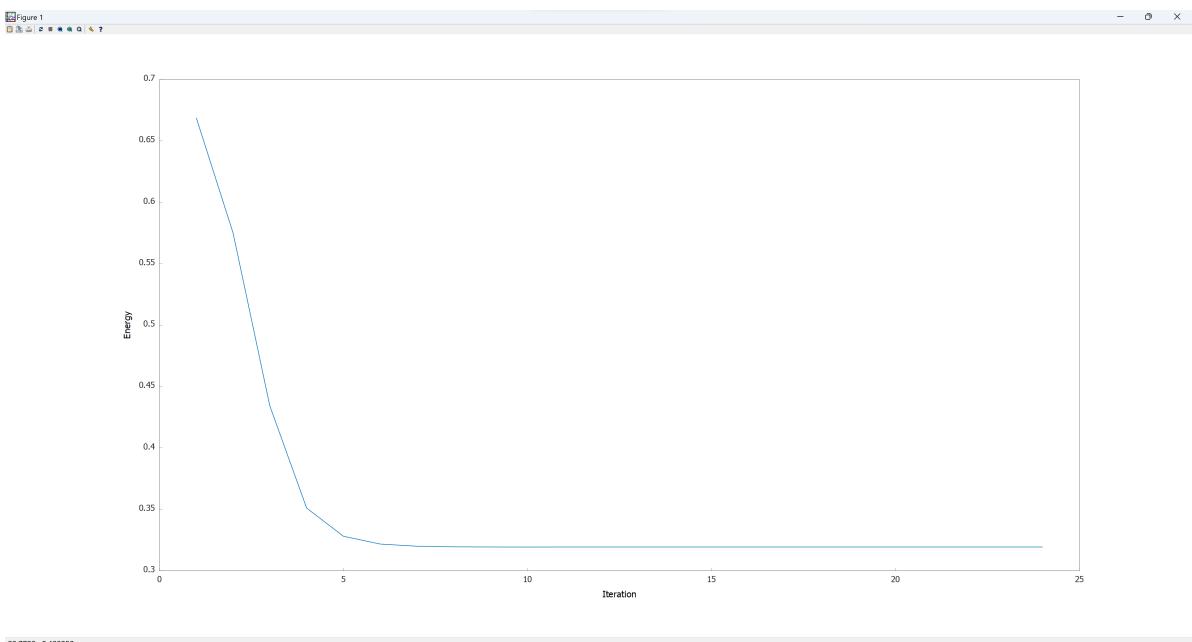
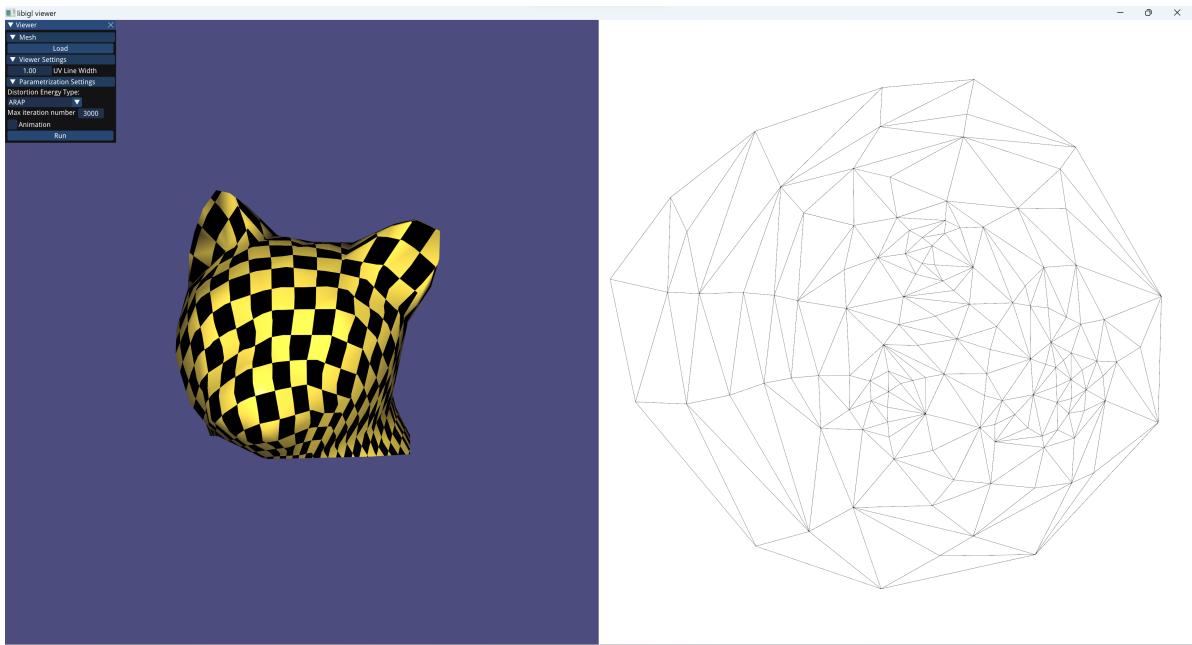
2077.41, 0.802351

cathead

对称Dirichlet能量



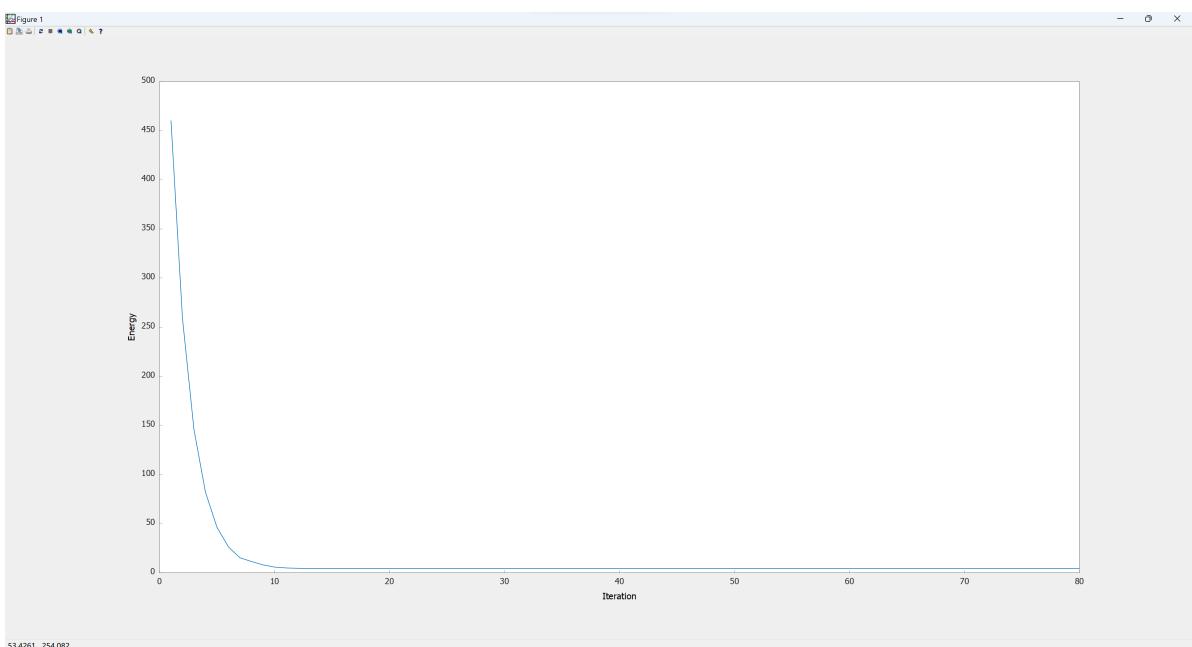
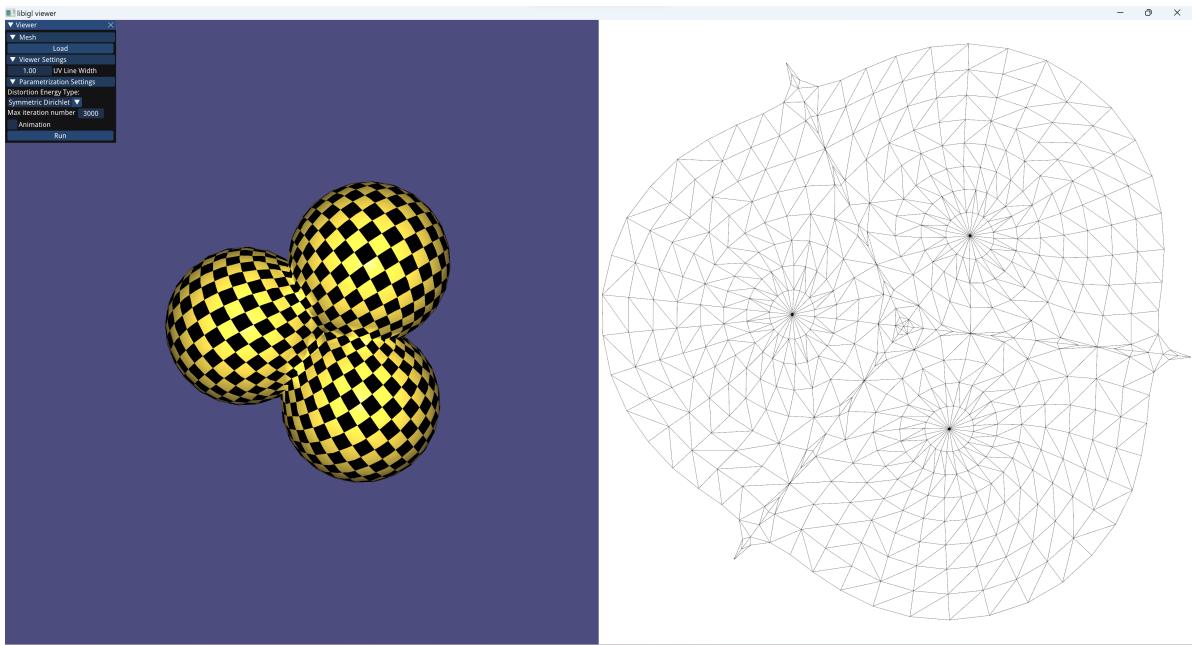
ARAP能量



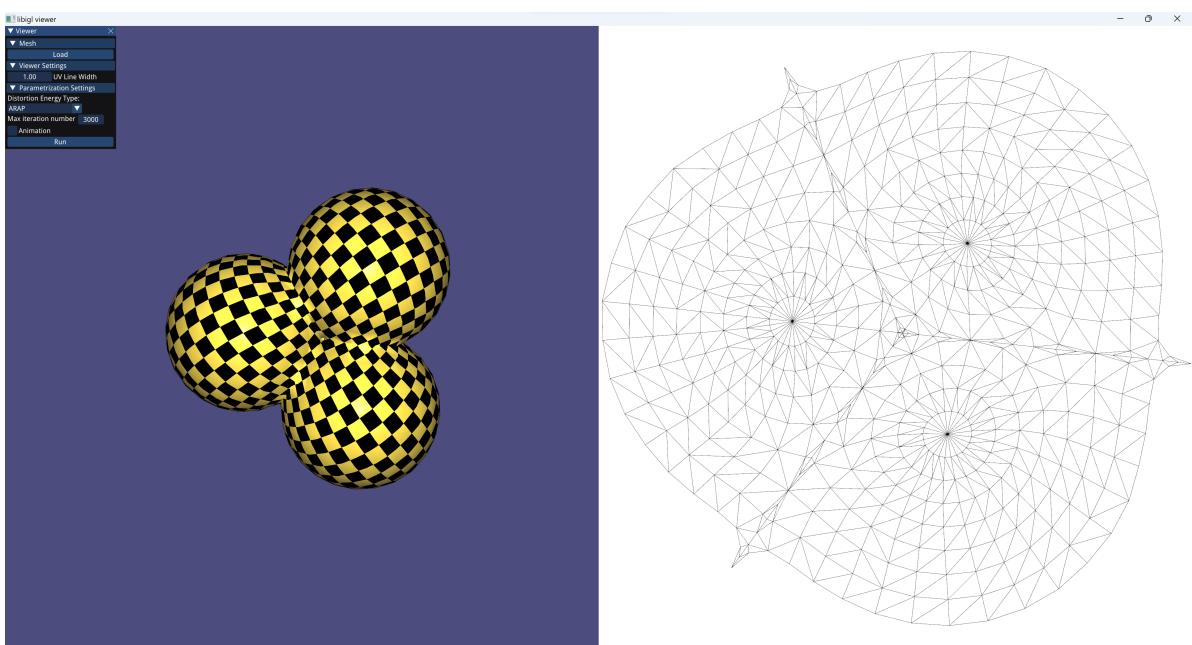
22.7792, 0.438952

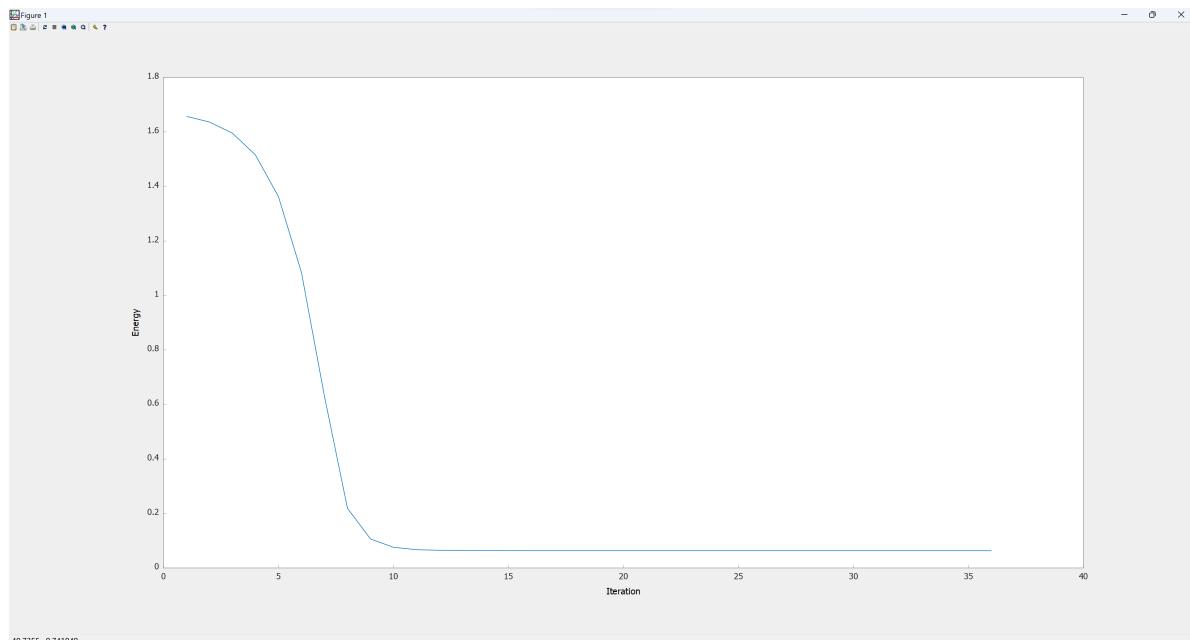
Balls

对称Dirichlet能量



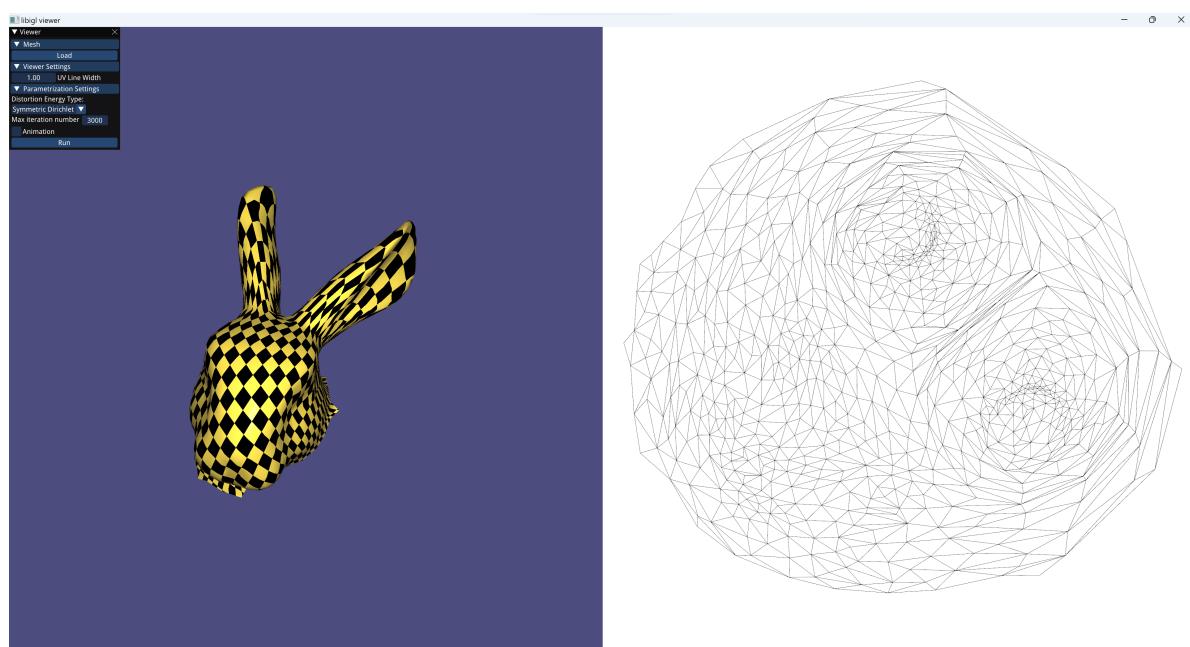
ARAP能量

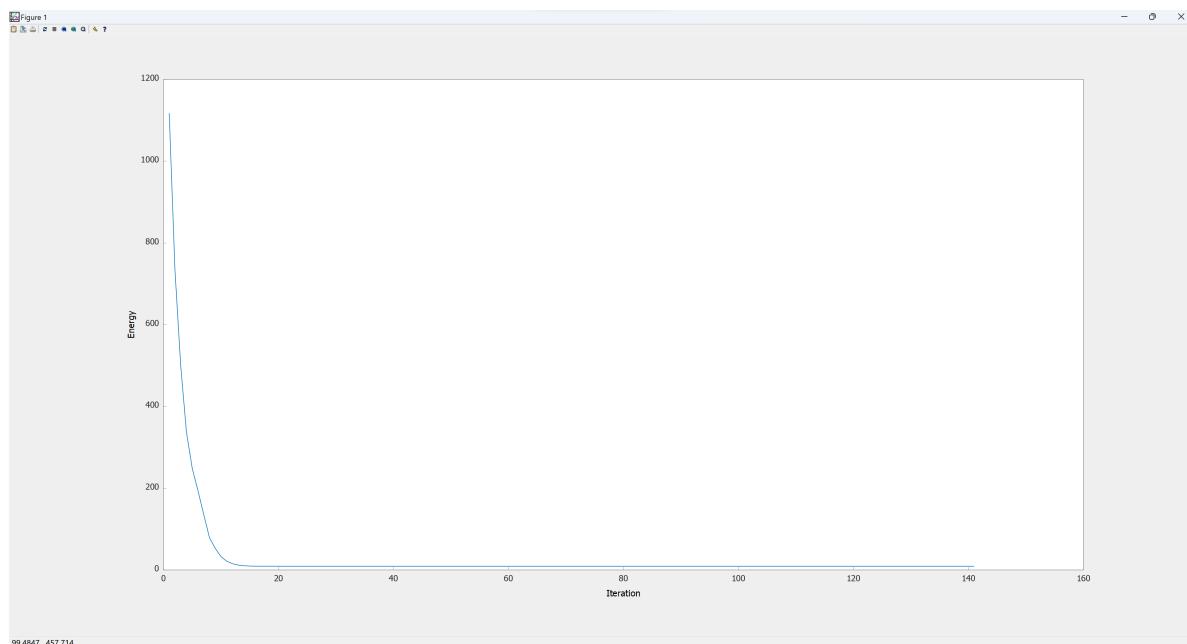




Bunny_head

对称Dirichlet能量





ARAP能量

