GAMES301 曲面参数化-HW4

Author: 彭博

Boundary First Flattening(BFF)是基于共形映射的曲面参数化算法。和LSCM相比BFF允许用户控制参数平面上曲面的边界，因此非常适合纹理映射这样对网格边界有一定需求的任务。BFF参数化结果可以直接运行`hw4.m`来进行查看。

Background

BFF算法涉及到比较多的数学推导，因此在介绍具体的实现前我们首先来整理一下相关的数学知识。

Confromal Maps

BFF的目标是计算一个流形到复平面的映射$f:\mathcal{M}\to\mathbb{C}$，当映射$f$能够保持切平面上向量的夹角时我们称这样的映射是一个**共形映射(confromal map)**。如果对于切平面上的任意向量X存在线性映射$\mathcal{J}$满足：

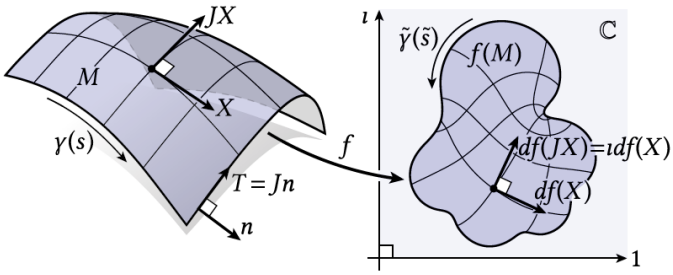
$$df(\mathcal{J}X)=i\,df(X)$$

则称f是一个**全纯映射(holomorphic map)**；其中$\mathcal{J}$称为f诱导的**共形结构(conformal structure)**，它相当于在切平面上将向量X逆时针旋转90°。实际上上式即为流形上的**Cauchy-Riemann方程(Cauchy-Riemann equation)**，如果df不会出现退化的情况则f一定是一个共形映射。

从度量的角度来看共形映射存在一个尺度缩放因子：

$$e^u=\frac{|df(X)|}{|X|}$$

e^u描述了切向量X在共形映射前后的缩放，且与X的方向无关；其中函数$u:\mathcal{M}\to\mathbb{R}$称为**对数共形因子(log conformal factor)**。



Conjugate Harmonic Function

共形映射的一个重要性质是它可以表示为一对**共轭调和函数(conjugate harmonic function)**。对于流形上的实值函数$a:\mathcal{M}\to\mathbb{R}$，当它满足Laplace方程时称其为调和函数：

$$\Delta a=0$$

其中Δ为**Laplace-Beltrami算子(Laplace-Beltrami operator)**。对于全纯映射$f=a+b\,i$，利用Cauchy-Riemann方程可以证明它的实部和虚部都满足Laplace方程：

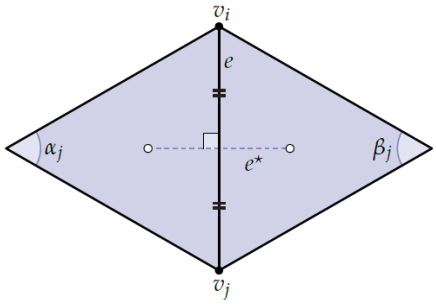
$$\Delta a=\nabla\cdot\nabla a=-\nabla\cdot(\mathcal{J}\nabla b)=0$$

$$\Delta b = \nabla \cdot \nabla b = \nabla \cdot (\mathcal{J} \nabla a) = 0$$

因此f实部和虚部的两个实值函数都是调和函数，称为一对共轭调和函数。

在离散网格上Laplace-Beltrami算子是一个矩阵$\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$，其中每个元素满足：

$$A_{ij} = \begin{cases} -\frac{1}{2}(\cot \alpha_j + \cot \beta_j), & \text{for each edge } ij \in E \\ -\sum_{ij \in E} A_{ij}, & \text{for each vertex } i \in V \end{cases}$$



Laplace-Beltrami算子的实现可以参考如下`cotLaplacian()`函数：

```
function L = cotLaplacian(V, F)
%% Build (sparse) cotangent-Laplacian matrix
%% Args:
%%      V[nV, 3]: vertices in 3D
%%      F[nF, 3]: face connectivity
%% Returns:
%%      L[nV, nV]: sparse cotangent-Laplacian matrix

nV = size(V, 1);

%% edges
Es = reshape(V(F(:, [2, 3, 1]), :) - V(F, :), [size(F), 3]);

%% trigonometry
coss =-dot(Es(:, [2, 3, 1], :), Es(:, [3, 1, 2], :), 3);
sins = vecnorm(cross(Es(:, [2, 3, 1], :), Es(:, [3, 1, 2], :), 3), 2, 3);
cots = coss ./ sins;

%% cotangent-Laplacian adjacency
L = sparse(F, F(:, [2, 3, 1]), cots, nV, nV);
L = 0.5*(L + L');

%% L = D - A
L = diag(sparse(sum(L, 1))) - L;

end
```

Poisson Equation

除了Laplace方程外，BFF还涉及到在网格上求解**Poisson方程(Poisson equation)**：

$$\Delta a = \phi$$

其中a和ϕ都是在网格上定义的实值函数。Poisson方程的解依赖于边界条件，当我们规定了a在边界上的函数值时称为**Dirichlet边界(Dirichlet condition)**：

$$\begin{cases}\Delta a = \phi, & \text{on } \mathcal{M} \\ a = g, & \text{on } \partial\mathcal{M}\end{cases}$$

而如果规定了a在边界上的梯度则称为**Neumann边界(Neumann condition)**：

$$\begin{cases}\Delta a = \phi, & \text{on } \mathcal{M} \\ \frac{\partial a}{\partial n} = h, & \text{on } \partial\mathcal{M}\end{cases}$$

在求解Poisson方程时可以通过在顶点dual cell上积分的方式来获得一系列线性方程组。我们将网格顶点分为内部顶点$\mathbf{I}$和边界顶点$\mathbf{B}$两部分，整理后可以得到：

$$\begin{bmatrix}\mathbf{A_{II}} & \mathbf{A_{IB}} \\ \mathbf{A_{BI}} & \mathbf{A_{BB}}\end{bmatrix}\begin{bmatrix}\mathbf{a_I} \\ \mathbf{a_B}\end{bmatrix} = \begin{bmatrix}\phi_\mathbf{I} \\ \phi_\mathbf{B}\end{bmatrix}$$

需要说明的是线性系统的右端并不是函数ϕ在顶点上的值，而是它在顶点dual cell上的积分。

对于Dirichlet边界相当于已知$\mathbf{a_B} = \mathbf{g} \in \mathbb{R}^{|\mathbf{B}|}$，因此只需要带入第一行进行求解即可：

$$\mathbf{A_{II}}\ \mathbf{a_I} = \phi_\mathbf{I} - \mathbf{A_{IB}}\ \mathbf{g}$$

而对于Neumann边界则需要对梯度在边界上进行积分，此时线性方程组转换为：

$$\begin{bmatrix}\mathbf{A_{II}} & \mathbf{A_{IB}} \\ \mathbf{A_{BI}} & \mathbf{A_{BB}}\end{bmatrix}\begin{bmatrix}\mathbf{a_I} \\ \mathbf{a_B}\end{bmatrix} = \begin{bmatrix}\phi_\mathbf{I} \\ \phi_\mathbf{B} - \mathbf{h}\end{bmatrix}$$

上式中$\mathbf{h} \in \mathbb{R}^{|\mathbf{B}|}$称为**discrete Neuman boundary data**，积分得到$\mathbf{h}$后只需按照通常的线性系统进行求解即可。

Yamabe Equation

Yamabe方程(Yamabe equation)刻画了共形映射曲率与对数共形因子之间的关系，在曲面的内部和边界上对数共形因子u满足：

$$\begin{cases}\Delta u = K - e^{2u}\tilde{K}, & \text{on } \mathcal{M} \\ \frac{\partial u}{\partial n} = \kappa - e^{u}\tilde{\kappa}, & \text{on } \partial\mathcal{M}\end{cases}$$

式中K和κ分别是原始曲面上的高斯曲率和测地曲率，而$\tilde{K}$和$\tilde{\kappa}$则是共形映射后参数平面上的高斯曲率及测地曲率。

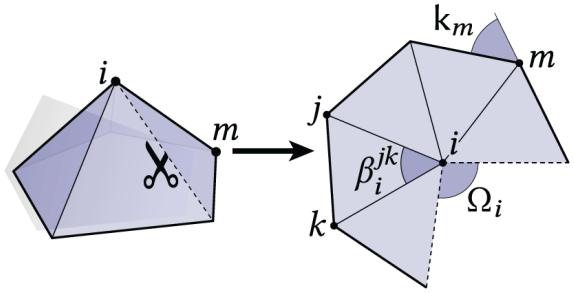
对于参数化问题，在参数平面上高斯曲率$\tilde{K}$恒为0。此时Yamabe方程表现为Poisson方程，通过在网格顶点的dual cell上进行积分可以得到：

$$\mathbf{A}u = \mathbf{\Omega}$$

式中$\mathbf{\Omega}$为内部顶点上的离散高斯曲率，它等于顶点相邻三角形的angle defect。

$$\Omega_i = 2\pi - \sum_{ijk \in F} \beta_i^{jk}$$

需要注意的是在网格的边界上高斯曲率为0。

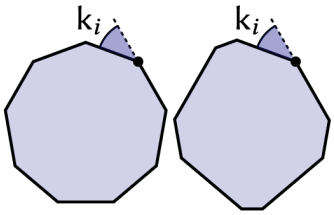


类似的，在网格边界的dual cell上进行积分可以得到Yamabe方程的边界条件：

$$\mathbf{h} = \mathbf{k} - \tilde{\mathbf{k}}$$

式中$\mathbf{k}$和$\tilde{\mathbf{k}}$分别是原始曲面边界和目标边界上的离散曲率，数值上等于把边界顶点相邻的三角形展平后的外角：

$$k_i = \pi - \sum_{ijk \in F} \beta_i^{jk}$$



计算网格上顶点高斯曲率以及边界测地曲率可以参考如下`discreteCurvature()`代码：

```
function [K, kappa] = discreteCurvature(V, F, B)
%% Compute discrete Gaussian and geodesic curvature
%% Args:
%%      V[nV, 3]: vertices in 3D
%%      F[nF, 3]: face connectivity
%%      B[1, nB]: boundary vertex index
%% Returns:
%%      K[nV, 1]: discrete Gaussian curvature
%%      kappa[nB, 1]: discrete geodesic curvature at the boundary

nV = size(V, 1);
nF = size(F, 1);

%% edges
Es    = reshape(V(F(:, [2, 3, 1]), :) - V(F, :), [size(F), 3]);
Enorm = vecnorm(Es, 2, 3);
Edir  = Es ./ Enorm;

%% interior angles
coss   =-dot(Edir(:, :, :), Edir(:, [3, 1, 2], :), 3);
angles = acos(coss);

%% sum over vertices
A = sparse(repmat((1:nF)', 1, 3), F, angles, nF, nV);
angles = full(sum(A, 1))';

%% discrete curvatures
```

```
K     = 2*pi - angles;
kappa = pi - angles(B);

%% set K=0 at the boundary
K(B) = 0;

end
```

从形式上来看，给定边界曲线的外角时求解Yamabe方程等价于求解具有Neumann边界$\mathbf{h}$的Poisson方程；而给定边界曲线的缩放因子情况下求解Yamabe方程等价于求解具有Dirichlet边界$u_{\mathbf{B}}$的Poisson方程。

需要额外说明的是前面介绍的Laplace算子是一个(半)正定算子，而在推导Yamabe方程时使用的是(半)负定的Laplace算子，二者实际上相差一个符号。由于半正定的Laplace算子在解方程时有更好的数学性质，BFF在求解Yamabe方程时对等式右端的高斯曲率项取了负号。

Curve Integration

通过求解Yamabe方程可以得到曲面在参数平面上的边界。需要注意的是在离散情况下边界曲线需要满足封闭条件，即曲线首尾的端点和切向相互重合。

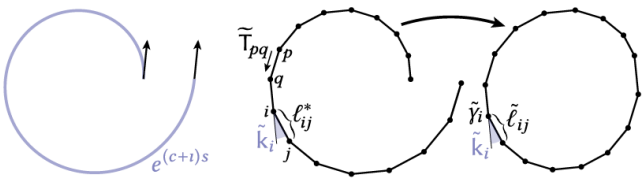
BFF通过构造一个优化问题来求解Yamabe方程给出的边界曲线。记原始曲面边界上端点ij构成的线段长为l_{ij}，映射后在参数平面上它的长度为：

$$l_{ij}^* = e^{\frac{u_i+u_j}{2}} l_{ij}$$

为了满足曲线的封闭性，我们将线段ij的长度调整为$\tilde{l}_{ij}$并构造约束优化问题：

$$\min_{\tilde{l}:\mathbf{B}\to\mathbb{R}} \frac{1}{2} \sum_{ij\in\partial\mathcal{M}} l_{ij}^{-1} |\tilde{l}_{ij} - l_{ij}^*|^2$$
$$\text{s.t.} \sum_{ij\in\partial\mathcal{M}} \tilde{l}_{ij}\tilde{T}_{ij} = 0$$

其中$\tilde{T}_{ij} = (\cos\varphi_i, \sin\varphi_i)$为线段$ij$在参数平面上的方向。



上述优化问题存在闭式解：

$$\tilde{l} = l^* - \mathbf{N}^{-1}\tilde{\mathbf{T}}^T(\tilde{\mathbf{T}}\mathbf{N}^{-1}\tilde{\mathbf{T}}^T)^{-1}\tilde{\mathbf{T}}\, l^*$$

其中$\mathbf{N} \in \mathbb{R}^{|\mathbf{B}|\times|\mathbf{B}|}$是一个对角矩阵，它的对角元素为顶点$i$对偶边长度的倒数$N_{ii} = \frac{1}{l_i}$；而$\tilde{\mathbf{T}} \in \mathbb{R}^{2\times|\mathbf{B}|}$则是线段切向列向量组成的矩阵。求解得到每一段线段的长度后进行累加就得到了参数平面上的边界曲线：

$$\tilde{\gamma}_p = \sum_{i=1}^{p-1} \tilde{l}_{ij}\tilde{T}_{ij}$$

计算边界曲线的过程可以参考如下`bestFitCurve()`函数。

```
function gamma = bestFitCurve(L, Ltarget, k)
```

```
%% Find the best fitted curve with given length and exterior angles
%% Args:
%%      L[nB, 1]: length of original curve segment
%%      Ltarget[nB, 1]: length of target curve segment
%%      k[nB, 1]: exterior angle of target curve segment
%% Returns:
%%      gamma[nB, 2]: fitted curve vertex

%% accumulate exterior angles and tangent vectors
phi = cumsum(k) - k(1);
T = [cos(phi) sin(phi)]';

%% boundary mass matrix
l = 0.5*(L + circshift(L, 1));     %% dual length
Ninv = diag(l);

%% solve optimal length to close the curve
TNTinv = matrixInv2x2(T * Ninv * T');
L = Ltarget - Ninv*T'*TNTinv*T*Ltarget;

%% accumulate scaled tangents
gamma = cumsum(L .* T', 1);
gamma = circshift(gamma, 1, 1);
gamma(1,:) = [0 0];

end
```

Poincaré-Steklov Operators

Poincaré-Steklov算子(Poincaré-Steklov operator)能够将微分方程的某种边界条件映射为同解的另一种边界条件。在BFF算法中我们需要2类Poincaré-Steklov算子分别处理Poisson方程和Cauchy-Riemann方程。

Dirichlet to Neumann

当我们已知曲面边界上的对数共形因子时求解Yamabe方程等价于求解具有Dirichlet边界的Poisson方程。为了获取边界上的曲率我们需要将Dirichlet边界转换为等价的Neumann边界，此时的Poincaré-Steklov算子可以表示为：

$$\mathbf{h} = \Lambda_\phi \mathbf{g} = \phi_\mathbf{B} - \mathbf{A}_\mathbf{IB}^T \mathbf{A}_\mathbf{II}^{-1} (\phi_\mathbf{I} - \mathbf{A}_\mathbf{IB}\ \mathbf{g}) - \mathbf{A}_\mathbf{BB}\ \mathbf{g}$$

不难发现Poincaré-Steklov算子Λ_ϕ相当于求解了一个Dirichlet边界的Poisson方程，然后在边界上计算源项$\phi_\mathbf{B}$与Poisson方程解的差异作为Neumann边界。

Neumann to Dirichlet

类似地，Neumann边界转换为Dirichlet边界的Poincaré-Steklov算子可以表示为：

$$\mathbf{g} = \Lambda_\phi^\dagger \mathbf{h} = \mathbf{a}_\mathbf{B}$$

我们只需要求解Neumann边界的Poisson方程，其边界上的解即为所需的Dirichlet边界。不过需要注意的是Neumann边界下Poisson方程不具有唯一解，不同的解之间相差一个常数。

Hilbert Transform

Hilbert变换(Hilbert transform)能够将全纯函数$f = a + b\,i$中实部a在边界上切向的方向导数映射为虚部b在边界法向上的导数。在离散情况下当我们已知实部a时，求解虚部b等价于求解具有Neumann边界的Laplace方程，边界$\mathbf{h}$可以表示为：

$$h_j = \frac{1}{2}(a_k - a_i)$$

Interpolation

BFF算法的最后一步是利用边界曲线$\tilde{\gamma}$来求解共轭调和方程得到参数化坐标。为了保证计算得到的映射是全纯的，我们需要分别考虑实部和虚部。对于实部a只需要求解具有Dirichlet边界的Laplace方程：

$$\Delta a = 0 \quad \text{s.t. } a|_{\partial\mathcal{M}} = \mathrm{Re}(\tilde{\gamma})$$

而对于虚部b则需要利用Hilbert变换将a转换为相应的Neumann边界：

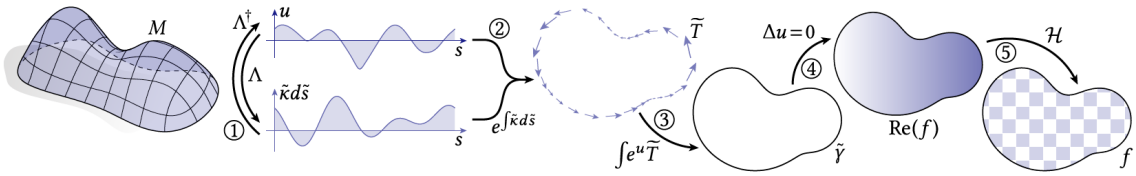
$$\Delta b = 0 \quad \text{s.t. } \frac{\partial b}{\partial n} = \mathcal{H}a$$

Implementation

BFF Framework

整个BFF参数化算法的流程如下：

1. 根据用户输入的边界信息通过求解Yamabe方程得到参数化后边界上的曲率(外角)$\tilde{\kappa}$和对数共形因子u：
   - 如果已知曲率$\tilde{\kappa}$则求解Neumann边界的Yamabe方程得到边界上的目标对数共形因子u；
   - 如果已知对数共形因子u则求解Dirichlet边界的Yamabe方程得到边界上的目标曲率$\tilde{\kappa}$；
2. 利用目标曲率$\tilde{\kappa}$和对数共形因子u构造一条封闭曲线$\tilde{\gamma}$作为参数化边界；
3. 求解共轭调和方程得到参数化坐标。



BFF with Given Exterior Angles

对于已知参数平面上边界曲线外角的情况可以使用如下`BFFAngle()`代码实现BFF参数化：

```
function uv = BFFAngle(V, F, k)
%% Boundary First Flattening with given exterior angles
%% Args:
%%      V[nV, 3]: vertices in 3D
%%      F[nF, 3]: face connectivity
%%      k[nB, 1]: target curvature at the boundary
%% Returns:
%%      uv[nV, 2]: uv coordinates

%% find boundary
[B, ~] = findBoundary(V, F);
```

```
nV = size(V, 1);

%% build Laplacian matrix
A = cotLaplacian(V, F);
A = A + 1e-8*speye(nV);

%% seperatate interior and boundary vertex
I = setdiff(1:nV, B);

AII = A(I,I);
AIB = A(I,B);
ABB = A(B,B);

%% discrete curvatures
[K, kappa] = discreteCurvature(V, F, B);

%% solve Yamabe equation (Neumann to Dirichlet)
phi = -K;
h = kappa - k;      %% Neumann data
phi(B) = phi(B) - h;

u = A \ phi;
u = u-mean(u);      %% constant offset

%% solve boundary curve length
uB = u(B);
uBr= circshift(uB, -1); Br = circshift(B, -1);     %% left shift the array to
find right endpoint
L  = vecnorm(V(B,:)-V(Br,:), 2, 2);
Ltarget  = exp((uB+uBr)/2) .* L;

%% best fit curve
gamma = bestFitCurve(L, Ltarget, k);

%% extend curve
uv = zeros(nV, 2);
uv(B, 1) = gamma(:, 1);
uv(I, 1) = AII \ (-AIB*gamma(:, 1));

%% Hilbert transform
aB = uv(B, 1); h = zeros(nV, 1);
h(B, :) =-0.5*(circshift(aB, -1) - circshift(aB, 1));
uv(:, 2) = A \ h;

uv = uv - mean(uv, 1);

end
```

BFF with Given Scaling Factors

如果已知边界曲线映射前后的对数共形因子则可以使用如下`BFFScale()`代码实现BFF参数化：

```
function uv = BFFScale(V, F, u)
%% Boundary First Flattening with given scale factors
%% Args:
%%      V[nV, 3]: vertices in 3D
%%      F[nF, 3]: face connectivity
%%      u[nB, 1]: target scale factor at the boundary
%% Returns:
%%      uv[nV, 2]: uv coordinates

nV = size(V, 1);

%% find boundary
[B, ~] = findBoundary(V, F);

%% build Laplacian matrix
A = cotLaplacian(V, F);
A = A + 1e-8*speye(nV);

%% seperatate interior and boundary vertex
I = setdiff(1:nV, B);

AII = A(I,I);
AIB = A(I,B);
ABB = A(B,B);

%% discrete curvatures
[K, kappa] = discreteCurvature(V, F, B);

%% solve Yamabe equation (Dirichlet to Neumann)
phi =-K;
a = AII \ (phi(I)-AIB*u);
k = kappa - (phi(B) - AIB'*a - ABB*u);
k = k / sum(k) * 2 * pi;

%% solve boundary curve length
uB = u;
uBr = circshift(uB, -1); Br = circshift(B, -1);     %% left shift the array to
find right endpoint
L  = vecnorm(V(B,:)-V(Br,:), 2, 2);
Ltarget  = exp((uB+uBr)/2) .* L;

%% best fit curve
gamma = bestFitCurve(L, Ltarget, k);

%% extend curve
uv = zeros(nV, 2);
uv(B, 1) = gamma(:, 1);
uv(I, 1) = AII \ (-AIB*gamma(:, 1));

%% Hilbert transform
aB = uv(B, 1); h = zeros(nV, 1);
```

```
h(B, :) =-0.5*(circshift(aB, -1) - circshift(aB, 1));
uv(:, 2) = A \ h;

uv = uv - mean(uv, 1);

end
```

Applications

我们还可以对`BFFAngle()`和`BFFScale()`两个函数进行定制来对适应不同的参数化需求。

Automatic Parameterization

如果用户对边界曲线没有偏好则可以将边界上的对数共形因子设置为0，此时BFF计算得到的参数坐标相当于最小化面积扭曲。我们定义`BFFAuto()`函数来实现这样的自动参数化：

```
function uv = BFFAuto(V, F)
%% Automatic parameterization with free boundary
%% Args:
%%      V[nV, 3]: vertices in 3D
%%      F[nF, 3]: face connectivity
%% Returns:
%%      uv[nV, 2]: uv coordinates

%% find boundary
[B, ~] = findBoundary(V, F);

u = zeros(length(B), 1);
uv = BFFScale(V, F, u);

end
```

Uniformization

根据**曲面单值化定理(uniformization theorem)**，亏格为0的开放曲面都拓扑同胚于复平面上的圆盘。因此可以构造一个共形映射将曲面参数化到一个圆盘上，且这个映射在圆盘边界上具有$\kappa = 1$的常曲率。

要实现这样的参数化可以将曲面边界上的目标曲率设置为1然后调用`BFFAngle()`，不在BFF论文指出这种做法往往只会得到一个凸的边界而不是圆盘边界。为了克服这样的问题论文中提出了一种迭代参数化方法，只需要每次令边界上的曲率正比于上一步参数化后对偶边的长度即可：

$$\tilde{k}_i^n \leftarrow \frac{2\pi\, \tilde{l}_i^{n-1}}{\sum_{i \in \mathbf{B}} \tilde{l}_i^{n-1}}$$

试验表明绝大多数情况下我们都可以在10次迭代之内收敛。圆盘参数化可以参考如下`BFFUniform()`代码：

```
function uv = BFFUniform(V, F)
%% Uniformization to a disk with BFF
%% Args:
%%      V[nV, 3]: vertices in 3D
%%      F[nF, 3]: face connectivity
%% Returns:
```

```
%%      uv[nV, 2]: uv coordinates

%% find boundary
[B, ~] = findBoundary(V, F);

nV = size(V, 1);
Br = circshift(B, -1);

for i=1:10
    %% dual edge length
    L = vecnorm(V(B,:)-V(Br,:), 2, 2);
    l = 0.5*(L + circshift(L, 1));

    %% set exterior angle proportional to the most recent dual lengths
    k = l / sum(l) * 2 * pi;

    uv = BFFAngle(V, F, k);
    V = [uv zeros(nV, 1)];
end

uv = uv(:, 1:2);

end
```

Flatten to Square

对于参数化到矩形区域这样的问题我们同样可以通过设置边界上的目标曲率来实现，不过此时BFF会得到带圆角的光滑边界曲线而不是具有尖锐转角的矩形边界。为了解决这个问题论文指出我们可以忽略Hilbert变换，通过直接求解两个调和方程来保持边界曲线的曲率，参见`BFFSquare()`函数。对于简单的边界曲线，这种做法得到的参数化是所需全纯映射的一个近似。

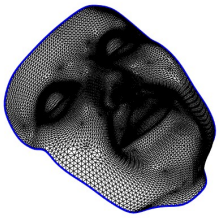
```
function uv = BFFSquare(V, F)
%% Uniformization to a square with BFF
%% Args:
%%      V[nV, 3]: vertices in 3D
%%      F[nF, 3]: face connectivity
%% Returns:
%%      uv[nV, 2]: uv coordinates

nV = size(V, 1);

%% find boundary
[B, ~] = findBoundary(V, F);

%% fix boundary exterior angle
nB= length(B);
k = zeros(nB,1);
k(floor((1:4)*nB/4)) = 2*pi/4;

%% build Laplacian matrix
A = cotLaplacian(V, F);
A = A + 1e-8*speye(nV);

%% seperatate interior and boundary vertex
I = setdiff(1:nV, B);
```

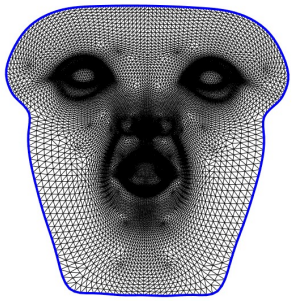
```
AII = A(I,I);
AIB = A(I,B);
ABB = A(B,B);

%% discrete curvatures
[K, kappa] = discreteCurvature(V, F, B);

%% solve Yamabe equation (Neumann to Dirichlet)
phi = -K;
h = kappa - k;      %% Neumann data
phi(B) = phi(B) - h;

u = A \ phi;
u = u-mean(u);      %% constant offset

%% solve boundary curve length
uB = u(B);
uBr= circshift(uB, -1); Br = circshift(B, -1);     %% left shift the array to
find right endpoint
L  = vecnorm(V(B,:)-V(Br,:), 2, 2);
Ltarget  = exp((uB+uBr)/2) .* L;

%% best fit curve
gamma = bestFitCurve(L, Ltarget, k);

%% extend curve with harmonics
uv = zeros(nV, 2);
uv(B, :) = gamma;
uv(I, :) = AII \ (-AIB*gamma);

uv = uv - mean(uv, 1);

end
```

Result

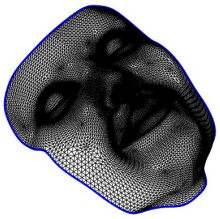
在`face.obj`网格上使用`BFFAuto()`、`BFFUniform()`、`BFFSquare()`3个函数可以得到不同的参数化结果如下：

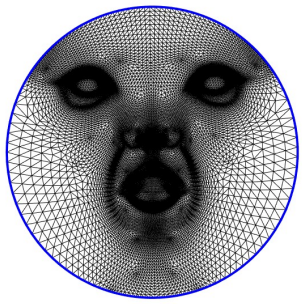
Automatic Parameterization



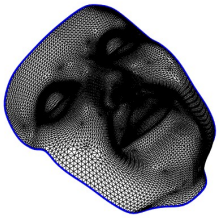


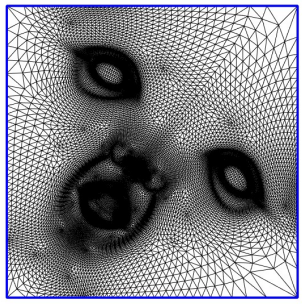
Uniformization





Flatten to Square





Reference

- Boundary First Flattening Project Homepage
- Boundary First Flattening Video