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 Tutte's Parameterization **1**

1.1 摘要 1

1.2 引言 1

1.3 表面参数化方法 1

1.3.1 图的定义及相关概念 1

1.3.2 曲面参数化 2

1.4 实验结果 4

1.4.1 三角形曲面的平面嵌入 5

1.4.2 纹理映射 7

1.5 总结 8

附录 A 代码修改说明 **9**

A.1 对原有文件的修改 9

A.2 新增文件 14

A.3 错误更正 20

第 1 章 Tutte's Parameterization

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。本次作业将实现 Tutte[1, 2] 提出的“barycentric mapping”（重心映射）图绘制方法和 Floater[3] 提出的“shape preserving”（保形）曲面参数化方法，并基于四组三角形曲面模型测试对应算法的表面参数化效果。

1.2 引言

现有的曲面参数化方法纷繁复杂。本实验报告实现的两组方法（Tutte Barycentric Mapping 和 Floater's Weight Parameterization）将帮助我们理解在上世纪 60 年代至 90 年代，众多学者在表面参数化方向的研究成果。

1.3 表面参数化方法

1.3.1 图的定义及相关概念

在此之前，我们需要给出图的定义：

定义 1.1 (图)

一个图结构可以表示为 $G = G(V, E)$，其中 $V = \{x_1, x_2, \cdots, x_N\}$ 是顶点集合，$E = \{(x_i, x_j) \mid x_i \text{ 和 } x_j \text{ 之间存在边}, i \neq j\}$ 是边集合。

♣

此外，我们可以给出部分基于图的定义 [4]：

定义 1.2

1. x_j 是 x_i 的邻居当且仅当存在边 $(x_i, x_j) \in E$。
2. x_i 的度 d_i 指 x_i 的不同的邻居数量。
3. 如果图 H 的顶点集合和边集合分别是图 G 的顶点集合和边集合的子集，则称图 H 是图 G 的子图。
4. 如果图 G 可以嵌入一个平面，且
   (a) 每个顶点 $x_i \in V$ 可以映射到 $P_i \in \mathbb{R}^2$，
   (b) 每条边 $(x_i, x_j) \in E$ 映射到一条以 P_i 和 P_j 为端点的曲线上，
   (c) 曲线之间只在公共端点处相交；
   那么称图 G 是平面图。

♣

我们可以描述三角形表面如下：

定义 1.3

一个平面三角化指简单联通的三角平面图，它的边是直线。

♣

令 F 表示面集合，则有

定义 1.4

一个三角形曲面 $S = S(G,X)$ 是指一个简单联通三角化平面图 $G(V,E,F)$，$V = \{x_1, x_2, \cdots, x_N\}$ 在 $\mathbb{R}^3$ 的嵌入。它的顶点集合为 $X = \{\boldsymbol{x}_i | \boldsymbol{x}_i \in \mathbb{R}^3\}$，边集合为直线，面为三角形面。 ♣

1.3.2 曲面参数化

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

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

令 $S(G,X)$，$G = G(V,E,F)$ 是一个三角化曲面，且顶点集合 $X = \{\boldsymbol{x}_i = (x_i, y_i, z_i)\}_{i=1}^N$。对顶点 $\boldsymbol{x}_i$ 重新编号，使得 $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ 是满足任意逆时针方向的边界点，而 $\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+2}, \cdots, \boldsymbol{x}_N$ 是内部点。令 $K = N - n$，考虑如下的过程：

1. 选择 $P_1, P_2, \cdots, P_n$ 作为边数为 K 的凸多边形 $D \in \mathbb{R}^2$ 的顶点，排列满足逆时针顺序。
2. 对于 $x_i \in V$，$i \in \{n+1, n+2, \cdots, N\}$，选择 $\lambda_{i,j}$ 满足

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

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

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

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

对于上述描述的过程，我们有如下的结论：

定理 1.1 (Tutte [2])

对于方程 1.2，当对于所有的边 $(x_i, x_j) \in E$ 选取 $\lambda_{i,j} = 1/d_i$ 时存在唯一解。 ♡

定理 1.2 (Floater [3])

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

$$\boldsymbol{A} = \begin{array}{c} \\ P_1 \\ \vdots \\ P_n \\ P_{n+1} \\ \vdots \\ P_N \end{array} \begin{array}{c} \begin{array}{cccccc} j & 1 & \cdots & n & n+1 & \cdots & N \end{array} \\ \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ \cdots & -\lambda_{n+1,j_p} & \cdots & 1 & -\lambda_{n+1,j_q} & \cdots \\ \cdots & \ddots & \ddots & \ddots & \ddots & \cdots \\ \cdots & -\lambda_{N,j_p} & \cdots & -\lambda_{N,j_q} & \cdots & 1 \end{bmatrix} \end{array} \tag{1.3}$$

是非奇异的。 ♡

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

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

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

基于上述的流程，Tutte 和 Floater 的曲面参数化方法均满足框架 1

Algorithm 1: Tutte Embedding 算法流程

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

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

1 计算 S 的边界 ∂G，根据结果对结点重新编号;

2 将正 n 边形的顶点坐标赋予 $b_{N\times 2}=[P_1,P_2,\cdots,P_n,0,\cdots,0]^T$;

3 **for** $k=n+1$ **to** N **do**

4 　基于 Tutte 或者 Floater 的方法计算 λ_{k,j_l}，$l\in\{1,2,\cdots,d_k\}$;

5 **end**

6 根据公式 1.3 计算 A 并求解方程 1.4 得到 $x_{N\times 2}=[P_1,P_2,\cdots,P_n,P_{n+1},\cdots,P_N]^T$;

7 输出 $\mathcal{P}=\mathcal{P}(G,U_b,\Lambda)$;

算法 1 的第 3 行代码中，计算 λ_{i,j_k} 的步骤是 Tutte 方法和 Floater 方法的不同之处，我们分别称之为“Tutte 权重”（averaged）和“Floater 权重”（shape-preserved）。Tutte 权重的选择方法是 $\lambda_{i,j_k}=1/d_i$，而 Floater [3] 权重的计算方法可以总结如算法 2.

Algorithm 2: Floater 权重计算

Data: 顶点 $\boldsymbol{x}_i$ 及邻居结点 $X_i = \{\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \cdots, \boldsymbol{x}_{j_{d_i}}\}$.

Result: $\boldsymbol{x}_{j_l}$ 对于 $\boldsymbol{x}_i$ 的权重 $\{\lambda_{i,j_1}, \lambda_{i,j_2}, \cdots, \lambda_{i,j_{d_i}}\}$.

初始化 $P = (0,0)$，$P_1 = (\|\boldsymbol{x}_{j_1} - \boldsymbol{x}_i\|, 0)$，并计算 $\theta_i = \sum_{k=1}^{d_i} \text{ang}(\boldsymbol{x}_{j_k}, \boldsymbol{x}_i, \boldsymbol{x}_{j+1})$;

for $k = 2$ **to** d_i **do**

令 $\text{ang}(\boldsymbol{x}_i, \boldsymbol{x}_0, \boldsymbol{x}_k) = \langle \overrightarrow{\boldsymbol{x}_0\boldsymbol{x}_i}, \overrightarrow{\boldsymbol{x}_0\boldsymbol{x}_k} \rangle$，$P_k$ 满足

$$\begin{cases} \|P_k - P\| = \|\boldsymbol{x}_{j_k} - \boldsymbol{x}_i\|, \\ \text{ang}(P_{k-1}, P, P_k) = \dfrac{2\pi}{\theta_i} \text{ang}(\boldsymbol{x}_{j_{k-1}}, \boldsymbol{x}_i, \boldsymbol{x}_{j_k}). \end{cases}$$

令 $\varphi_{k+1} = \text{ang}(P_k, P, P_{k+1})$，$\psi_{k+1} = \text{ang}(\boldsymbol{x}_k, \boldsymbol{x}_i, \boldsymbol{x}_{k+1})$. 计算 P_{k+1} 如下：

$$\begin{bmatrix} P_{k+1}.x \\ P_{k+1}.y \end{bmatrix} = \begin{bmatrix} s_{k+1} & 0 \\ 0 & s_{k+1} \end{bmatrix} \begin{bmatrix} \cos\varphi_{k+1} & -\sin\varphi_{k+1} \\ \sin\varphi_{k+1} & \cos\varphi_{k+1} \end{bmatrix} \begin{bmatrix} P_k.x \\ P_k.y \end{bmatrix}$$

其中

$$s_{k+1} = \frac{\|\boldsymbol{x}_{j_{k+1}} - \boldsymbol{x}_i\|}{\|\boldsymbol{x}_{j_k} - \boldsymbol{x}_i\|} \quad \text{并且} \quad \varphi_{k+1} = \frac{2\pi}{\theta_i}\psi_{k+1}.$$

end

for $l = 1$ **to** d_i **do**

计算 $P_{r(l)}$ 和 $P_{r(l)+1}$，其中 $P \in \triangle P_l P_{r(l)} P_{r(l)+1}$ 并且直线 PP_l 与线段 $P_{r(l)}P_{r(l)+1}$ 相交;

求解 $P = \delta_1 P_l + \delta_2 P_{r(l)} + \delta_3 P_{r(l)+1}$ 如下：

$$\begin{bmatrix} P_l.x & P_{r(l)}.x & P_{r(l)+1}.x \\ P_l.y & P_{r(l)}.y & P_{r(l)+1}.y \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

各自将 $\mu_{l,l} = \delta_1$, $\mu_{r(l),l} = \delta_2$ 和 $\mu_{r(l)+1,l} = \delta_3$ 加到顶点 P_l, $P_{r(l)}$ 和 $P_{r(l)+1}$ 的权重上;

计算

$$\lambda_{i,j_k} = \frac{1}{d_i} \sum_{l=1}^{d_i} \mu_{k,l}, \quad k = 1, \cdots, d.$$

输出 $\{\lambda_{i,j_1}, \lambda_{i,j_2}, \cdots, \lambda_{i,j_{d_i}}\}$;

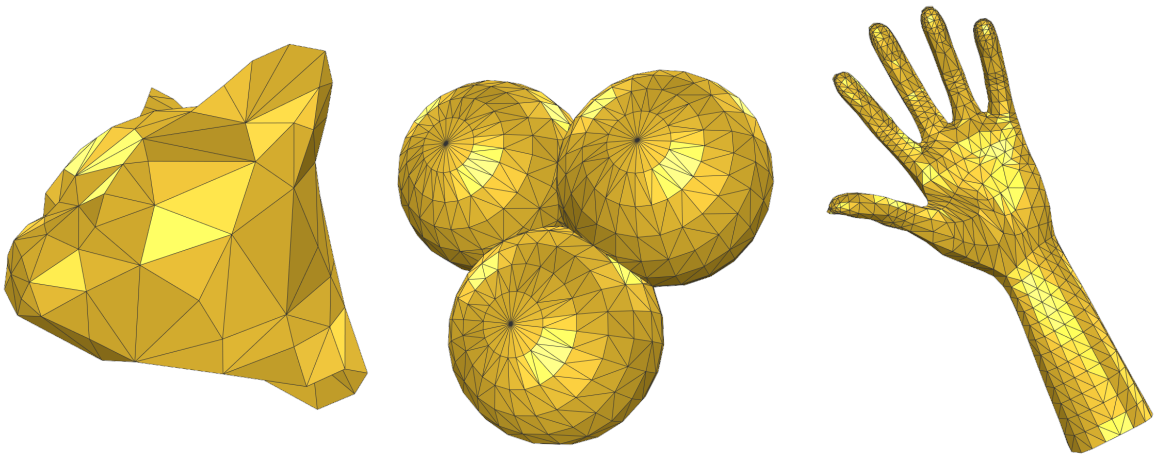
end

1.4 实验结果

我们将采用三组数据进行测试，如表 1.1。对应的模型绘制结果如图 1.1。

名称	格式	面类型	V	E	F	边界数	存储
cathead	OBJ	三角面	131	378	248	1	8KB
Balls	OBJ	三角面	547	1578	1032	1	26KB
hand	OFF	三角面	1558	4653	3096	1	97KB

表 1.1: 测试数据

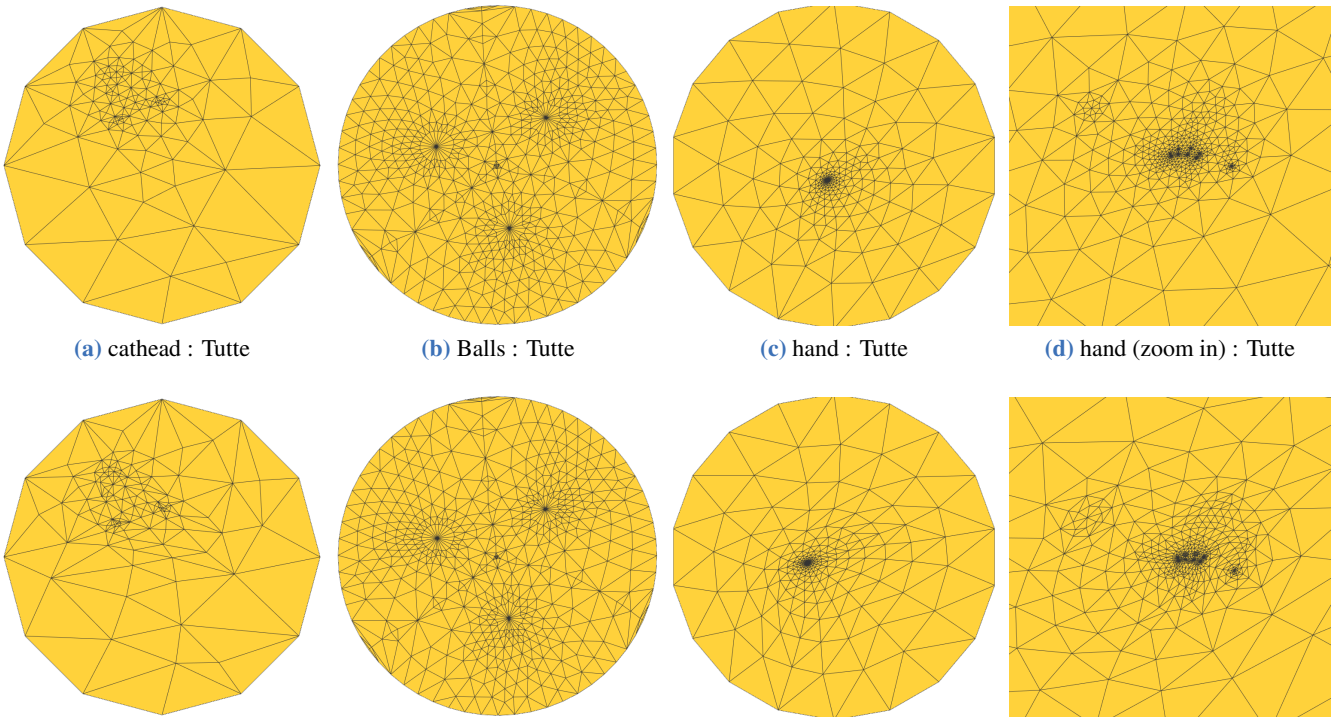


(a) cathead.obj (b) Balls.obj (c) hand.off

图 1.1: 测试数据渲染图（扁平着色）

1.4.1 三角形曲面的平面嵌入

此章节添加了不同多边形边界的参数化实验，包括与选取的边界边数相同的正多边形（图 1.2）、三角形（图 1.3）、正方形（图 1.4）和正五边形（图 1.5）。需要注意的是，多边形的边数不会超过被选取的三角形曲面边界的边数。从图 1.2～ 图 1.5的结果可以看出，基于 Tutte Embedding（算法 1）得到的参数化结果都呈现出“与边界点距离越远的顶点越集中”的趋势。

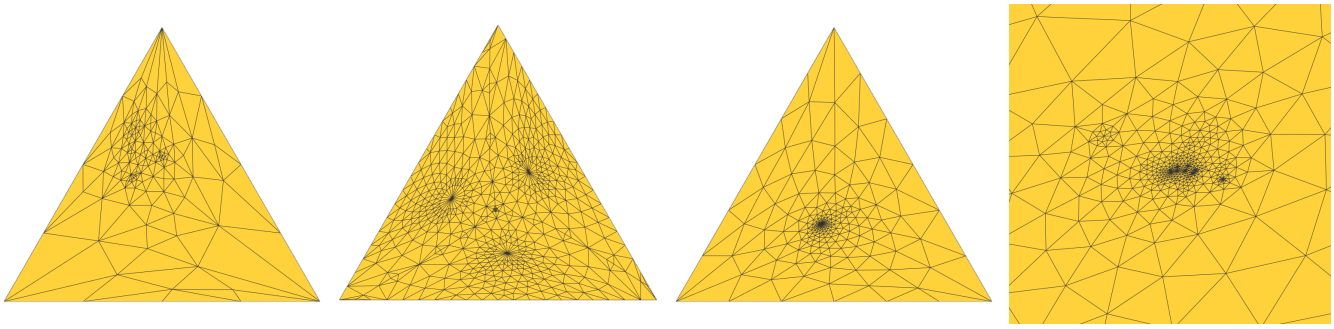


(a) cathead : Tutte (b) Balls : Tutte (c) hand : Tutte (d) hand (zoom in) : Tutte

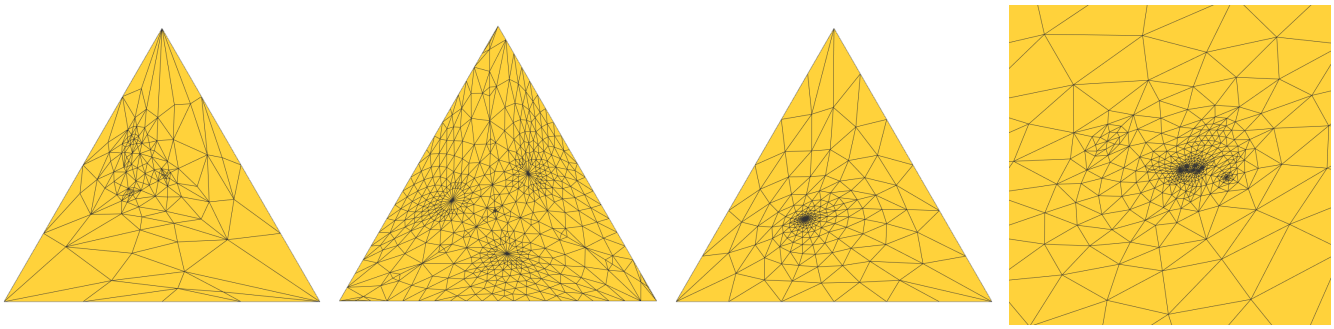
(e) cathead : Floater (f) Balls : Floater (g) hand : Floater (h) hand (zoom in) : Floater

图 1.2: 测试数据渲染图（边界形状：正多边形）

此外，根据参数化结果可以看出，二维凸多边形的边界并未对参数化结果的拓扑结构造成显著影响，更多的是影响曲面参数和下一章节中纹理映射的结果。显然太过集中的顶点会导致纹理坐标被极

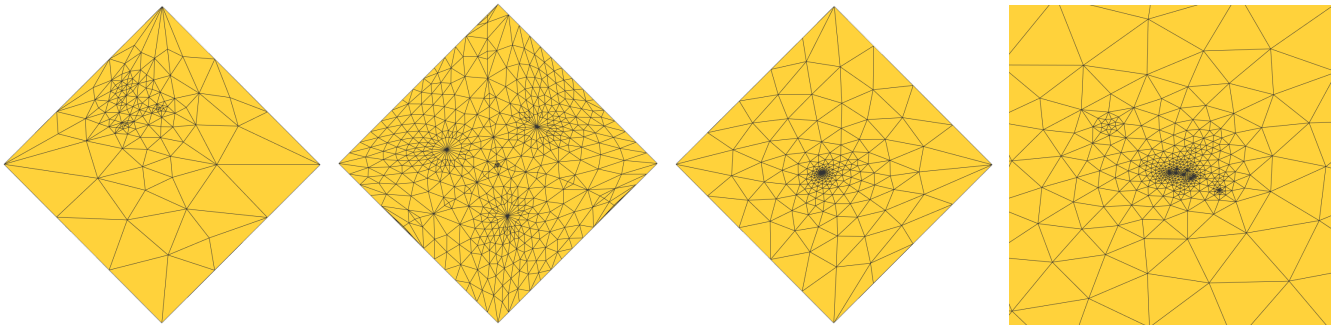


(a) cathead : Tutte (b) Balls : Tutte (c) hand : Tutte (d) hand (zoom in) : Tutte

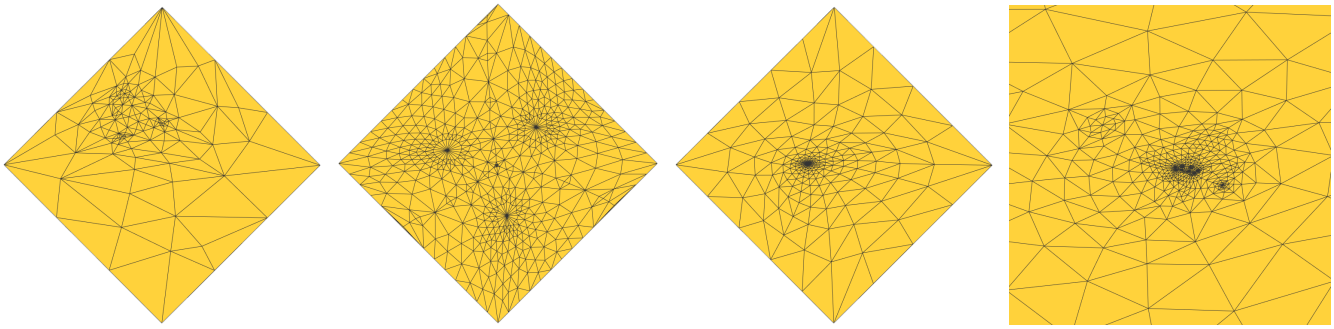


(e) cathead : Floater (f) Balls : Floater (g) hand : Floater (h) hand (zoom in) : Floater

图 1.3: 测试数据渲染图（边界形状：正三角形）



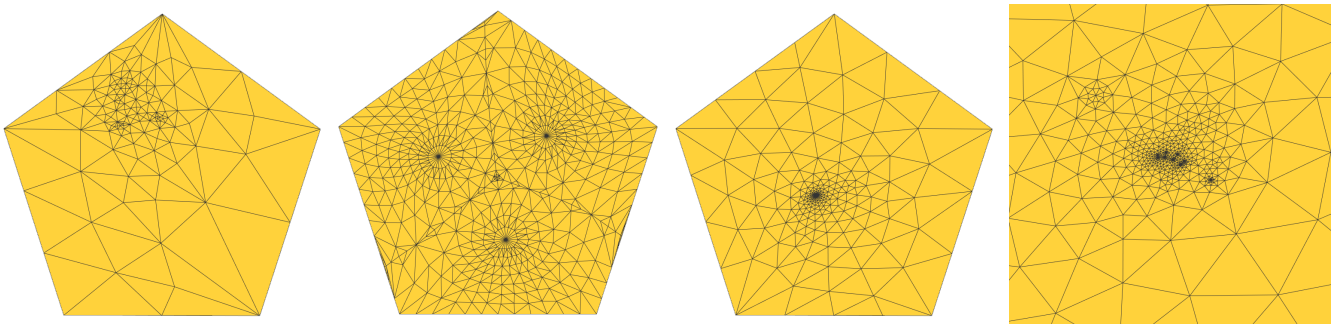
(a) cathead : Tutte (b) Balls : Tutte (c) hand : Tutte (d) hand (zoom in) : Tutte



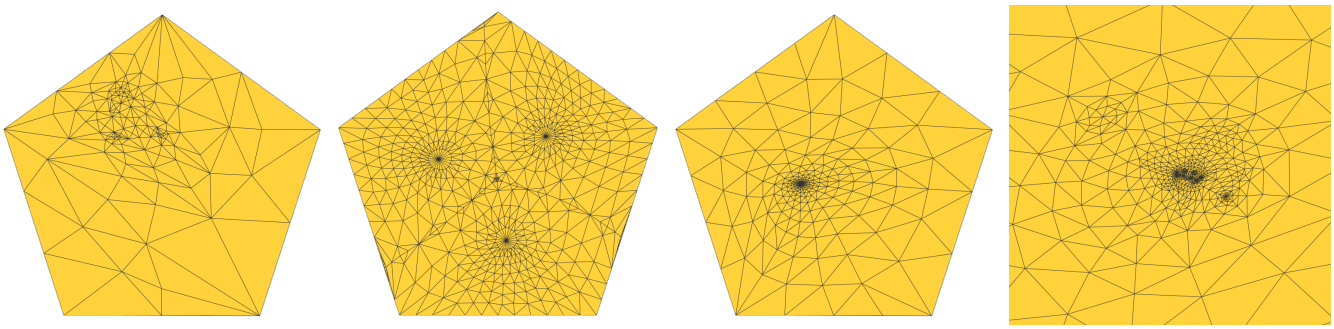
(e) cathead : Floater (f) Balls : Floater (g) hand : Floater (h) hand (zoom in) : Floater

图 1.4: 测试数据渲染图（边界形状：正四边形）

大地扭曲，在下一节的纹理映射中可以观察到这种现象。



(a) cathead : Tutte (b) Balls : Tutte (c) hand : Tutte (d) hand (zoom in) : Tutte

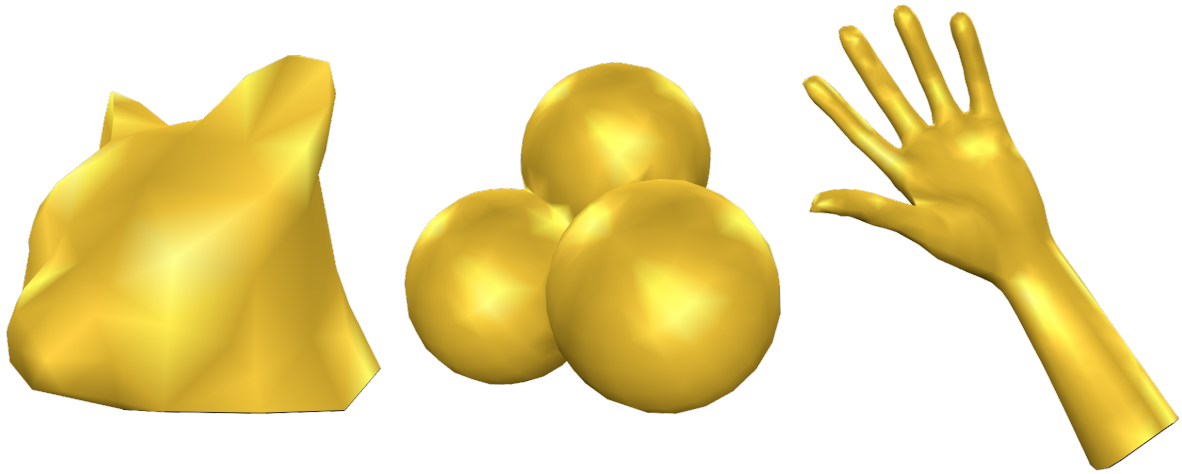


(e) cathead : Floater (f) Balls : Floater (g) hand : Floater (h) hand (zoom in) : Floater

图 1.5: 测试数据渲染图（边界形状：正五边形）

1.4.2 纹理映射

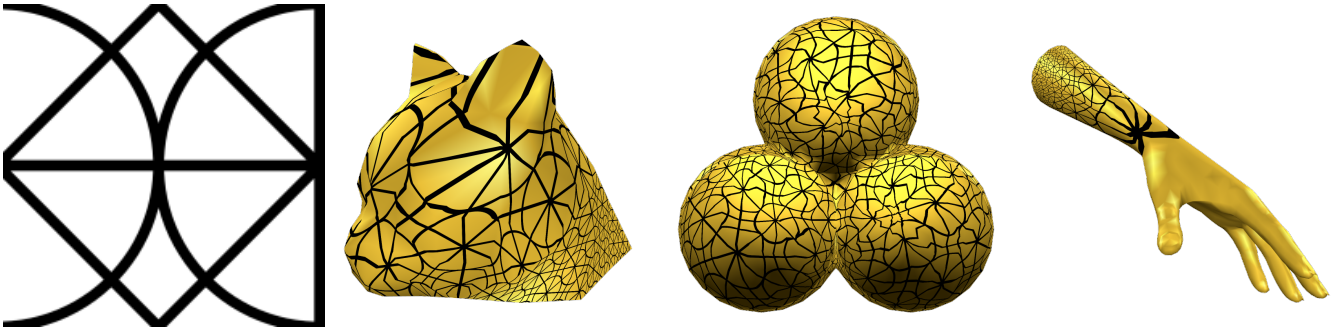
本文选取的参数化结果中范围是 $P_i \in [0,1]^2$，以便于我们进行纹理映射。在此之前，需要修改平直着色模式至平滑着色模式（图 1.6）。此外，我们采用两种 UV 映射贴图，以凸显 Tutte 方法和 Floater 方法的不同。需要注意的是，纹理贴图和几何模型的线框模型遮挡严重，这里不放出参数化多边形的 UV 贴图及对应线框下的贴图效果。



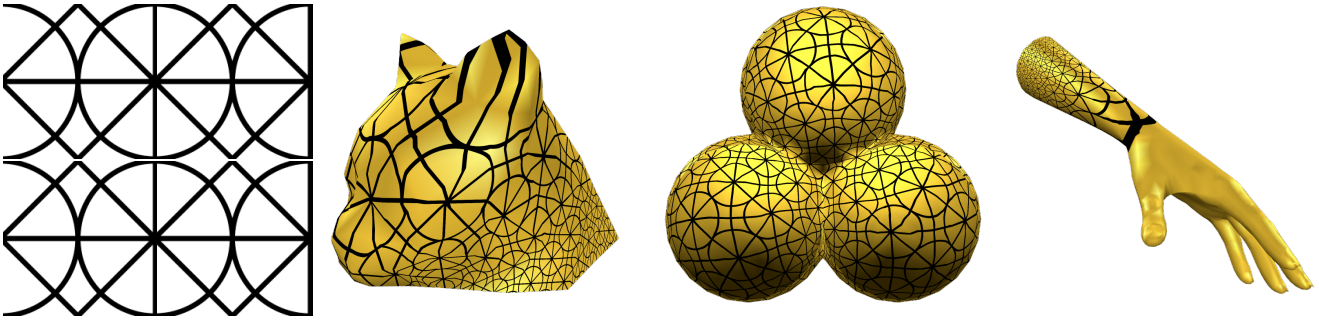
(a) cathead.obj (b) Balls.obj (c) hand.off

图 1.6: 测试数据渲染图（平滑着色）

具体在两种实验贴图的效果如图 1.7 和图 1.8。可以看出，Tutte 的方法并不能保证贴图和模型表面的一致性，会出现贴图毛刺、抖动等问题；而 Floater 方法有效地避免了这些问题。

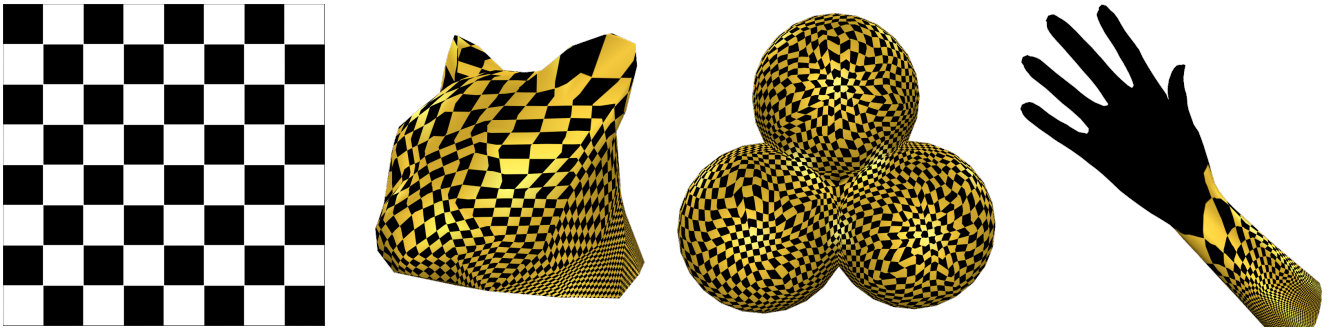


(a) pi-factory pattern (b) cathead : Tutte (c) Balls : Tutte (d) hand : Tutte

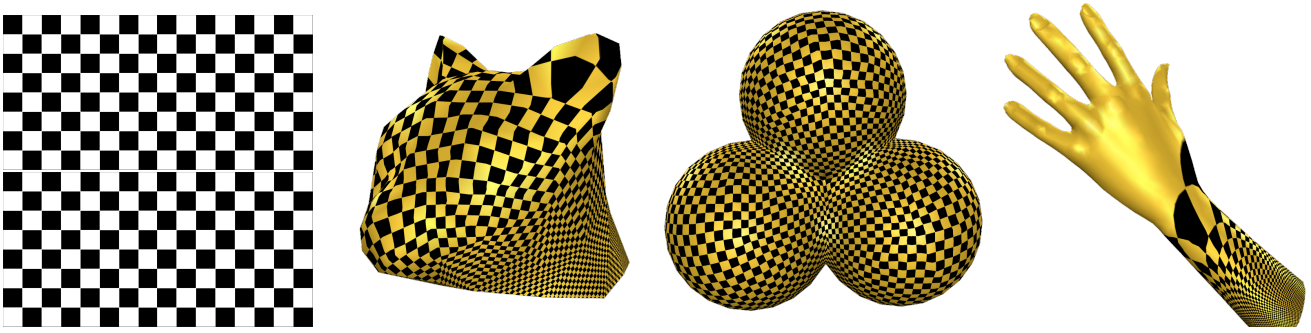


(e) pi-factory pattern tiling (f) cathead : Floater (g) Balls : Floater (h) hand : Floater

图 1.7: 测试数据纹理映射渲染图（边界形状：正多边形）



(a) checkerboard pattern (b) cathead : Tutte (c) Balls : Tutte (d) hand : Tutte



(e) checkerboard pattern tiling (f) cathead : Floater (g) Balls : Floater (h) hand : Floater

图 1.8: 测试数据纹理映射渲染图（边界形状：正多边形）

1.5 总结

Tutte's Embedding 方法提供了一个无翻转的曲面参数化结果。囿于自身的缺陷，该方法生成的参数化结果不尽人意。此方法为我们提供了参数化的基本思路，并为后续的参数化工作打下了坚实基础。

附录 A 代码修改说明

A.1 对原有文件的修改

Listing A.1: surfacemeshprocessing.cpp

```
// ========== surfacemeshprocessing.h ==========
// add QAction for smooth rendering
// ...
QAction *actSmooth;
// ...
// add QAction for parameterization
QAction *actParamAverage;
QAction *actParamFloater;
// ...

// ========== surfacemeshprocessing.cpp ==========
void SurfaceMeshProcessing::CreateActions(void)
{
    // ...
    // smooth shading
    actSmooth = new QAction(tr("Smooth"), this);
    actSmooth->setIcon(QIcon(":/SurfaceMeshProcessing/Images/smooth.png"));
    actSmooth->setStatusTip(tr("Show smooth"));
    actSmooth->setCheckable(true);
    connect(actSmooth, SIGNAL(triggered()), viewer, SLOT(ShowSmooth()));
    // ...
    agViewGroup->addAction(actSmooth);
    // ...
    // parameterization action
    actParamAverage = new QAction("Average Weighted", this);
    actParamAverage->setStatusTip(tr("Neighbors Weighted Averagely"));
    connect(actParamAverage, SIGNAL(triggered()), viewer, SLOT(TutteParam_AverageWeight
    ()));

    actParamFloater = new QAction("Floater Weighted", this);
    actParamFloater->setStatusTip(tr("Neighbors Weighted Floaterly"));
    connect(actParamFloater, SIGNAL(triggered()), viewer, SLOT(TutteParam_FlaterWeight
    ()));
    // ...
}

void SurfaceMeshProcessing::CreateMenus(void)
{
    // ...
    QMenu* menuTools = menuBar()->addMenu(tr("&Tool"));
    QMenu* tutteParam = menuTools->addMenu(tr("Tutte's Param"));
```

```
    tutteParam->addAction(actParamAverage);
    tutteParam->addAction(actParamFloater);
    // ...
}

void SurfaceMeshProcessing::CreateToolBars(void)
{
    // ...
    tbView->addAction(actSmooth);
    // ...
}
```

Listing A.2: MainViewerWidget

```
// ========== MainViewerWidget.h ==========
class MainViewerWidget : public QDialog
{
    // ...
    // Tutte's parameterization API
    void TutteParam_AverageWeight();
    void TutteParam_FlaterWeight();
    // ...
}

// ========== MainViewerWidget.cpp ==========
void MainViewerWidget::TutteParam_AverageWeight()
{
    meshviewerwidget->TutteParam(MeshViewerWidget::TutteParamType::AVERAGE_WEIGHTED);
}

void MainViewerWidget::TutteParam_FlaterWeight()
{
    meshviewerwidget->TutteParam(MeshViewerWidget::TutteParamType::FLOATER_WEIGHTED);
}
```

Listing A.3: MeshViewerWidget

```
// ========== MeshViewerWidget.h ==========
class MeshViewerWidget : public QGLViewerWidget
{
    // ...
    // parameterization enums
    enum class TutteParamType { AVERAGE_WEIGHTED, FLOATER_WEIGHTED };
    // parameterization functions
    void TutteParam(TutteParamType type);
    // ...
    void DrawSmooth() const;
    // ...
    // Tutte embedding auxiliary function
```

```
  std::vector<double> CalAdjectWeight(acamcad::polymesh::MVert* v,
    const std::vector<acamcad::polymesh::MVert*>& adjVerts,
    MeshViewerWidget::TutteParamType type);
    ...
}

// ========== MeshViewerWidget.cpp ==========

// smooth shading mode
void MeshViewerWidget::DrawSmooth() const
{
  glShadeModel(GL_SMOOTH);

  glBindTexture(GL_TEXTURE_2D, glTextureID);
  glEnable(GL_TEXTURE_2D);

  glBegin(GL_TRIANGLES);

  for (const auto& fh : polyMesh->polyfaces())
  {
    for (const auto& fvh : polyMesh->polygonVertices(fh))
    {
      glNormal3dv(fvh->normal().data());
      auto uv = fvh->getTextureUVW().uv;
      float uvScale = 10.0f;
      glTexCoord2f(uv[0] * uvScale, uv[1] * uvScale);
      glVertex3dv(fvh->position().data());
    }
  }

  glEnd();
}

/// ========== my parameterization functions ==========
std::vector<double> MeshViewerWidget::CalAdjectWeight(acamcad::polymesh::MVert* v,
const std::vector<acamcad::polymesh::MVert*>& adjVerts,
MeshViewerWidget::TutteParamType type)
{
std::vector<double> weights;

switch (type)
{
case TutteParamType::AVERAGE_WEIGHTED:
    tutte::AverageParam(v, adjVerts, weights);
    break;
case TutteParamType::FLOATER_WEIGHTED:
    tutte::FloaterParam(v, adjVerts, weights);
    break;
}
```

```
62 return weights;
63 }
64
65 void MeshViewerWidget::TutteParam(TutteParamType type)
66 {
67   using mat = Eigen::MatrixXd;
68   using acamcad::polymesh::MVert;
69   std::cout << "Tutte's Parameterization\n";
70
71   if (polyMesh->numVertices() == 0)
72   {
73     std::cerr << "ERROR: TutteParam() No vertices!" << std::endl;
74     return;
75   }
76
77   /// ====== calculate boundary vertices ======
78   auto boundaryVertLists = polyMesh->boundaryVertices();
79   std::cout << "Boundary Points [" << boundaryVertLists.size() << "]\n";
80
81   // 1. prepare convex polygon
82   int M = boundaryVertLists.size();
83   int N = polyMesh->numVertices();
84
85   // TODO : Better to add a function to control the boundary polygon, or a slider?
86   auto boundaryUVs = tutte::GetBoundaryUVs(M, tutte::UVBoundaryType::POLYGON_CIRCLE);
87   //auto boundaryUVs = tutte::GetBoundaryUVs(M, tutte::UVBoundaryType::POLYGON_TRIANGLE
       );
88   //auto boundaryUVs = tutte::GetBoundaryUVs(M, tutte::UVBoundaryType::POLYGON_SQUARE);
89   //auto boundaryUVs = tutte::GetBoundaryUVs(M, tutte::UVBoundaryType::POLYGON_PENTAGON
       );
90
91   // 2. prepare matrix (Eigen::Dense)
92   mat A = mat::Zero(N, N);
93   mat x = mat::Zero(N, 2);
94   mat b = mat::Zero(N, 2);
95
96   // a) boundary vertices
97   for (int i = 0; i < M; ++i)
98   {
99     int vertID = boundaryVertLists[i]->index();
100
101    A(vertID, vertID) = 1.0;
102    b(vertID, 0) = boundaryUVs[i].x;
103    b(vertID, 1) = boundaryUVs[i].y;
104  }
105
106  // b) inner vertices
107  std::unordered_set<MVert*> boundaryVertsDict(boundaryVertLists.begin(),
       boundaryVertLists.end());
```

```

  for (int i = 0; i < N; ++i)
  {
    auto pVert = polyMesh->vert(i);
    if (boundaryVertsDict.count(pVert)) continue;

    int vID = pVert->index();

    auto adjVerts = polyMesh->vertAdjacentVertices(pVert);
    auto weights = CalAdjectWeight(pVert, adjVerts, type);
    double weightSUM = std::accumulate(weights.begin(), weights.end(), 0.0);
    for (int j = 0; j < adjVerts.size(); ++j) {
      auto adjID = adjVerts[j]->index();
      A(vID, adjID) = weights[j];
      A(vID, vID) = -weightSUM;
    }
  }

  /// 3. solve the equation Ax = b
  x = A.lu().solve(b);

  std::cout << "Tutte's parameterization finished, updating mesh ... \n";

  for (int i = 0; i < N; ++i)
  {
    auto pVert = polyMesh->vert(i);
    auto vID = pVert->index();

    // TODO : Better to add a function to set vertex position
    //pVert->setPosition(x(vID, 0), x(vID, 1), 0.0);
    pVert->setTexture(x(vID, 0), x(vID, 1));
  }

  std::cout << "Done\n";

  UpdateMesh();
  update();
}
```

Listing A.4: QGLViewerWidget

```
// ========== QGLViewerWidget.h ==========
class QGLViewerWidget : public QOpenGLWidget
{
    // ...
    // load uv textures
    void LoadTexture();
}

```

```cpp
// ========== QGLViewerWidget.cpp ==========
void QGLViewerWidget::LoadTexture()
{
    glGenTextures(1, &glTextureID);
    glBindTexture(GL_TEXTURE_2D, glTextureID);

    // load and generate texture
    int width, height, nChannels;
    unsigned char* data = stbi_load("../src/Images/tiling patterns/R-C3.jpg", &width, &height, &nChannels, 0);
    //unsigned char* data = stbi_load("../src/Images/tiling patterns/circle.jpg", &width, &height, &nChannels, 0);
    //unsigned char* data = stbi_load("../src/Images/tiling patterns/pie-factory.png", &width, &height, &nChannels, 0);
    //unsigned char* data = stbi_load("../src/Images/tiling patterns/cross.jpg", &width, &height, &nChannels, 0);

    GLint internalFormat = GL_RGBA;
    if (nChannels == 1) internalFormat = GL_RED;
    if (nChannels == 2) internalFormat = GL_RG;
    if (nChannels == 3) internalFormat = GL_RGB;
    if (nChannels == 4) internalFormat = GL_RGBA;

    if (data)    glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, width, height, 0, internalFormat, GL_UNSIGNED_BYTE, data);
    else
    {
        std::cout << "Failed to load texture" << std::endl;
        assert(false);
    }

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    stbi_image_free(data);
}
```

A.2 新增文件

文件 `stb_image.h` 和 `sub_image.cpp`，用于加载图像。

文件 `Util_TutteEmbedding.h` 和 `Util_TutteEmbedding.cpp`，（代码A.5），用于存放相关代码。

图片 用于添加纹理映射：

(a) src/Images/circle.jpg

(b) src/Images/cross.jpg

(c) src/Images/pie-factory.png

(d) src/Images/pie-factory_s.png

(e) src/Images/pie-factory_t.png

(f) src/Images/R-C.jpg

(g) src/Images/R-C3.jpg

Listing A.5: Util_TutteEmbedding

```
// ========== Util_TutteEmbedding.h ==========
#include <vector>
#include <glm\glm.hpp>

#include "../PolyMesh/include/PolyMesh/PolyMesh.h"

namespace tutte
{
  enum class UVBoundaryType {
    POLYGON_CIRCLE,
    POLYGON_TRIANGLE,
    POLYGON_SQUARE,
    POLYGON_PENTAGON,
    // TODO : uncomment this to implement more complex boundaries, for later experiments
    // POLYGON_STAR,
    // POLYGON_CROSS
  };

  std::vector<glm::dvec2> GetBoundaryUVs(size_t numVerts, UVBoundaryType type);

  bool IsOinTriangle(glm::dvec2 P1, glm::dvec2 P2, glm::dvec2 P3);

  auto FloaterParam_I_a(acamcad::polymesh::MVert* v,
    const std::vector<acamcad::polymesh::MVert*>& adjVerts);
  auto FloaterParam_I_b(double thetaSum, const std::vector<glm::dvec2>& neighborAngleInfo);
  auto FloaterParam_II(const std::vector<glm::dvec2>& adjUVs);

  void AverageParam(acamcad::polymesh::MVert* v,
    const std::vector<acamcad::polymesh::MVert*>& adjVerts,
    std::vector<double>& weights);

  void FloaterParam(acamcad::polymesh::MVert* v,
    const std::vector<acamcad::polymesh::MVert*>& adjVerts,
    std::vector<double>& weights);
}

```

```cpp
// ========== Util_TutteEmbedding.cpp ==========
#include "Util_TutteEmbedding.h"

#include <algorithm>
#include <numeric>

#include <Eigen3\Eigen>

namespace tutte
{
  std::vector<glm::dvec2> GetBoundaryUVs(size_t numVerts, UVBoundaryType type)
  {
    const double CIRCLE_RADIUS = 0.5;
    std::vector<glm::dvec2> boundaryUVs(numVerts, { CIRCLE_RADIUS, CIRCLE_RADIUS });

    // 1. polygon edges
    size_t polygonEdges = 3;
    switch (type)
    {
    case UVBoundaryType::POLYGON_CIRCLE:  polygonEdges = numVerts;  break;
    case UVBoundaryType::POLYGON_TRIANGLE:  polygonEdges = 3;     break;
    case UVBoundaryType::POLYGON_SQUARE:  polygonEdges = 4;     break;
    case UVBoundaryType::POLYGON_PENTAGON:  polygonEdges = 5;     break;
    }

    polygonEdges = std::min(polygonEdges, numVerts);

    // 2. initialization
    std::vector<int> polygonEdgePoints(polygonEdges, 0);
    {
      int V = numVerts;
      int i = 0;
      while (V-- > 0)
      {
        polygonEdgePoints[i]++;
        i = (i + 1) % polygonEdges;
      }
    }
    assert(std::accumulate(polygonEdgePoints.begin(), polygonEdgePoints.end(), 0) ==
    numVerts);

    double angleGap = 2.0 * M_PI / static_cast<double>(polygonEdges);
    double edgeLength = 2.0 * CIRCLE_RADIUS * glm::sin(angleGap / 2.0);

    // rotate matrix
    glm::dmat2x2 polygonRotateMat = glm::dmat2({ 0, 1 }, { -1, 0 });

    // 3. calculation
    auto boundaryUVIter = boundaryUVs.begin();
```

```
85      for (size_t edgeIdx = 0; edgeIdx < polygonEdges; ++edgeIdx)
86      {
87        glm::dvec2 basePoint = CIRCLE_RADIUS * glm::dvec2{ glm::cos(edgeIdx * angleGap),
        glm::sin(edgeIdx * angleGap) };
88        glm::dvec2 edgeDirection = CIRCLE_RADIUS * glm::dvec2(glm::cos(((edgeIdx + 1) %
        polygonEdges) * angleGap),
89            glm::sin(((edgeIdx + 1) % polygonEdges) * angleGap)) - basePoint;
90
91        int edgePoints = polygonEdgePoints[edgeIdx];
92        double edgeDisDelta = edgeLength / static_cast<double>(edgePoints);
93
94        int i = 0;
95        while (i++ < edgePoints) {
96          glm::dvec2 pointUV = basePoint + (i / static_cast<double>(edgePoints)) *
        edgeDirection;
97            *boundaryUVIter += (polygonRotateMat * pointUV);
98            boundaryUVIter++;
99        }
100     }
101
102     return boundaryUVs;
103   }
104
105   auto FloaterParam_I_a(acamcad::polymesh::MVert* v,
106     const std::vector<acamcad::polymesh::MVert*>& adjVerts)
107   {
108     double thetaI{ 0.0 };
109     int N = adjVerts.size();
110
111     std::vector<glm::dvec2> disANDangles(N, { 0.0, 0.0 });
112
113     for (int i = 0; i < N; ++i)
114     {
115       int j = (i + 1) % N;
116       auto vecI = adjVerts[i]->position() - v->position();
117       auto vecJ = adjVerts[j]->position() - v->position();
118
119       glm::dvec3 vi(vecI.x(), vecI.y(), vecI.z());
120       glm::dvec3 vj(vecJ.x(), vecJ.y(), vecJ.z());
121
122       double di = glm::sqrt(glm::dot(vi, vi));
123       double dj = glm::sqrt(glm::dot(vj, vj));
124
125       disANDangles[i].x = di;
126       disANDangles[i].y = glm::acos(glm::dot(vi, vj) / (di * dj));
127       thetaI += disANDangles[i].y;
128     }
129     assert(thetaI > 0);
130     return std::make_tuple(thetaI, disANDangles);
```

```
131 }
132
133 auto FloaterParam_I_b(double thetaSum, const std::vector<glm::dvec2>&
      neighborAngleInfo)
134 {
135   int N = neighborAngleInfo.size();
136
137   std::vector<glm::dvec2> adjUVs(N, { 0.0, 0.0 });
138   adjUVs[0] = { neighborAngleInfo[0].x, 0.0 };
139
140   for (int j = 1; j < N; ++j)
141   {
142     // 1. rotate factor
143     double thetaI = neighborAngleInfo[j - 1].y * (2.0 * M_PI / thetaSum);
144     double cosThetaI = glm::cos(thetaI);
145     double sinThetaI = glm::sin(thetaI);
146     glm::dmat2x2 rotateMat2({ cosThetaI, sinThetaI }, { -sinThetaI, cosThetaI });
147
148     // 2. scale factor
149     double scale = neighborAngleInfo[j].x / neighborAngleInfo[j - 1].x;
150     glm::dmat2x2 scaleMat2({ scale, 0.0 }, { 0.0, scale });
151
152     adjUVs[j] = scaleMat2 * rotateMat2 * adjUVs[j - 1];
153   }
154
155   return adjUVs;
156 }
157
158 // If triangle P1P2P3 centering the original point (0, 0)
159 bool IsOinTriangle(glm::dvec2 P1, glm::dvec2 P2, glm::dvec2 P3)
160 {
161   double t1 = P1.x * P2.y - P1.y * P2.x;
162   double t2 = P2.x * P3.y - P2.y * P3.x;
163   double t3 = P3.x * P1.y - P3.y * P1.x;
164
165   return t1 * t2 ≥ 0 && t1 * t3 ≥ 0 && t2 * t3 ≥ 0;
166 }
167
168 auto FloaterParam_II(const std::vector<glm::dvec2>& adjUVs)
169 {
170   int N = adjUVs.size();
171   std::vector<double> mu_ls(N, 0.0);
172
173   for (int i = 0; i < N; ++i) {
174
175     auto Pl = adjUVs.begin() + i;
176
177     auto Pr = adjUVs.begin() + (i + 1) % N;
178     auto Pn = adjUVs.begin() + (i + 2) % N;
```

```
179
180       auto IterAscend = [&]() {
181         Pr++;
182         Pn++;
183         if (Pr == adjUVs.end()) Pr = adjUVs.begin();
184         if (Pn == adjUVs.end()) Pn = adjUVs.begin();
185       };
186
187       while (Pn != Pl)
188       {
189         if (!IsOinTriangle(*Pl, *Pr, *Pn))
190         {
191           IterAscend();
192           continue;
193         };
194
195         Eigen::Matrix3d A;
196         A(0, 0) = Pl->x;
197         A(1, 0) = Pl->y;
198         A(2, 0) = 1.0;
199         A(0, 1) = Pr->x;
200         A(1, 1) = Pr->y;
201         A(2, 1) = 1.0;
202         A(0, 2) = Pn->x;
203         A(1, 2) = Pn->y;
204         A(2, 2) = 1.0;
205         Eigen::Vector3d b(0.0, 0.0, 1.0);
206
207         auto x = A.lu().solve(b);
208
209         mu_ls[Pl - adjUVs.begin()] += x(0);
210         mu_ls[Pr - adjUVs.begin()] += x(1);
211         mu_ls[Pn - adjUVs.begin()] += x(2);
212         mu_ls[i] += x(0, 0);
213
214         assert(!std::isnan(x(0) * x(1) * x(2)));
215
216         IterAscend();
217       }
218     }
219
220     std::for_each(mu_ls.begin(), mu_ls.end(), [N](double& v) { return v / static_cast<
      double>(N); });
221     return mu_ls;
222   }
223
224   void AverageParam(acamcad::polymesh::MVert* v,
225     const std::vector<acamcad::polymesh::MVert*>& adjVerts,
226     std::vector<double>& weights)
```

```
227   {
228     weights.clear();
229     weights.resize(adjVerts.size(), 1.0);
230   }
231
232   void FloaterParam(acamcad::polymesh::MVert* v,
233     const std::vector<acamcad::polymesh::MVert*>& adjVerts,
234     std::vector<double>& weights)
235   {
236     int N = adjVerts.size();
237     weights.clear();
238
239     /// ====== Stage 1 : Initialize (u, v) list ======
240     // a) thetaI
241     auto [thetaI, disANDangles] = FloaterParam_I_a(v, adjVerts);
242
243     // b) UVs
244     auto UVs = FloaterParam_I_b(thetaI, disANDangles);
245
246     /// ====== Stage 2 : Calculate lambda_[i, jk] ======
247     weights = FloaterParam_II(UVs);
248   }
249 }
```

A.3 错误更正

感谢 QQ 群的小伙伴们为我找出实现方法和代码中的错误，在此罗列对报告内容和代码的修改：

错误更正 更正了算法 2中第 3 行计算 s_{k+1} 的错误，更正之前为

$$s_{k+1} = \|P_i - P\| \cdot \frac{\|\boldsymbol{x}_{j_{k+1}} - \boldsymbol{x}_i\|}{\|\boldsymbol{x}_{j_k} - \boldsymbol{x}_i\|}.$$

在此感谢 **forty-twoo**（QQ 1013417276）和 **is 度假版**（QQ 1353321741）两位同学指出的错误。

错误更正 根据修改后的算法 2更正了代码，删除了冗余行（请查看代码 A.5）

```
double dp = scale * glm::sqrt(glm::dot(adjUVs[j - 1] - P, adjUVs[j - 1] - P));
```

在此感谢 **forty-twoo**（QQ 1013417276）同学指出代码中的错误。

方法改进 算法 2 中关于 P_i 的计算存在累计误差，但在当前小规模简单模型的测试算例下尚未出现问题。解决方案是计算旋转角度的前缀和，从而消除累计误差。此方法将应用在后续的改进当中。在此感谢**古明地绿**（QQ 746530916）同学提出的改进策略。

目录结构 目录结构进行了修改，添加了文件夹 `Surface_Framework_Cmake/src/homeworks/TutteEmbedding/`。

代码结构 代码结构进行了修改，将所有与 Tutte Embedding 相关的辅助函数转移到 `Surface_Framework_Cmake/src/homeworks/TutteEmbedding` 文件夹下的 `Util_TutteEmbedding.h` 和 `Util_TutteEmbedding.cpp` 文件中。

Bibliography

[1] William Thomas Tutte. “Convex representations of graphs”. In: *Proceedings of the London Mathematical Society* 3.1 (1960), pp. 304–320.

[2] William Thomas Tutte. “How to draw a graph”. In: *Proceedings of the London Mathematical Society* 3.1 (1963), pp. 743–767.

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

[4] Wai-Kai Chen. *Applied graph theory*. Vol. 13. Elsevier, 2012.