



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: 0.01

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	10
A.1 对原有文件的修改	10
A.2 新增文件	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 $V = \{x_1, x_2, \dots, x_N\}$ 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 $G(V, E, F)$, $V = \{x_1, x_2, \dots, x_N\}$ 在 \mathbb{R}^3 的嵌入。它的顶点集合为 $X = \{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^3\}$, 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：

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

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

1. 选择 P_1, P_2, \dots, 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 $x_i \in V$, $i \in \{n + 1, n + 2, \dots, N\}$, 选择 $\lambda_{i,j}$ 满足

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

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

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

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

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, $\mathbf{x}_{j_1}, \mathbf{x}_{j_2}, \dots, \mathbf{x}_{j_{d_i}}$ 是满足逆时针顺序的邻接结点, 那么在公式 1.1 的选取方式下, 矩阵

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

是非奇异的。



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

$$\begin{array}{ccccccccc} j & 1 & \cdots & n & n+1 & \cdots & N & u & v \\ P_1 & \left[\begin{array}{cccccc} 1 & \cdots & 0 & 0 & \cdots & 0 \end{array} \right] & \left[\begin{array}{cc} x_0 & y_0 \\ \vdots & \vdots \\ x_n & y_n \\ x_{n+1} & y_{n+1} \\ \vdots & \vdots \\ x_N & y_N \end{array} \right] & = & \left[\begin{array}{cc} x & y \\ P_1.x & P_1.y \\ \vdots & \vdots \\ P_n.x & P_n.y \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{array} \right]. \end{array} \quad (1.4)$$

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

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

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

1 计算 S 的边界 ∂G , 根据结果对结点重新编号;

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

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

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

5 **end**

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

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

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 \mathbf{x}_i 及邻居结点 $X_i = \{\mathbf{x}_{j_1}, \mathbf{x}_{j_2}, \dots, \mathbf{x}_{j_{d_i}}\}$.

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

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

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

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

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

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

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

4 **end**

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

6 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;

7 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;

8 计算

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

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

9 **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

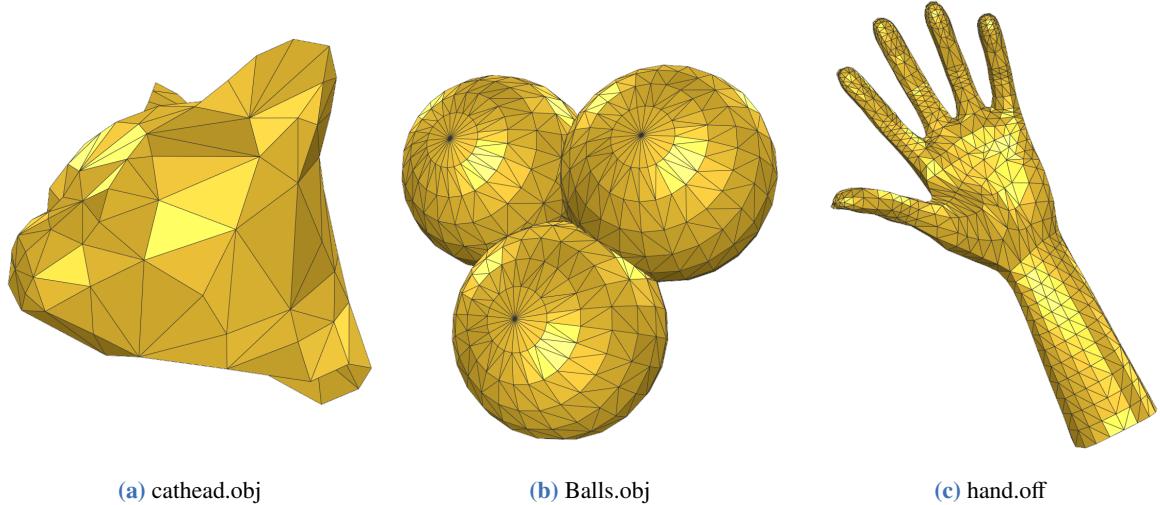


图 1.1: 测试数据渲染图 (扁平着色)

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。

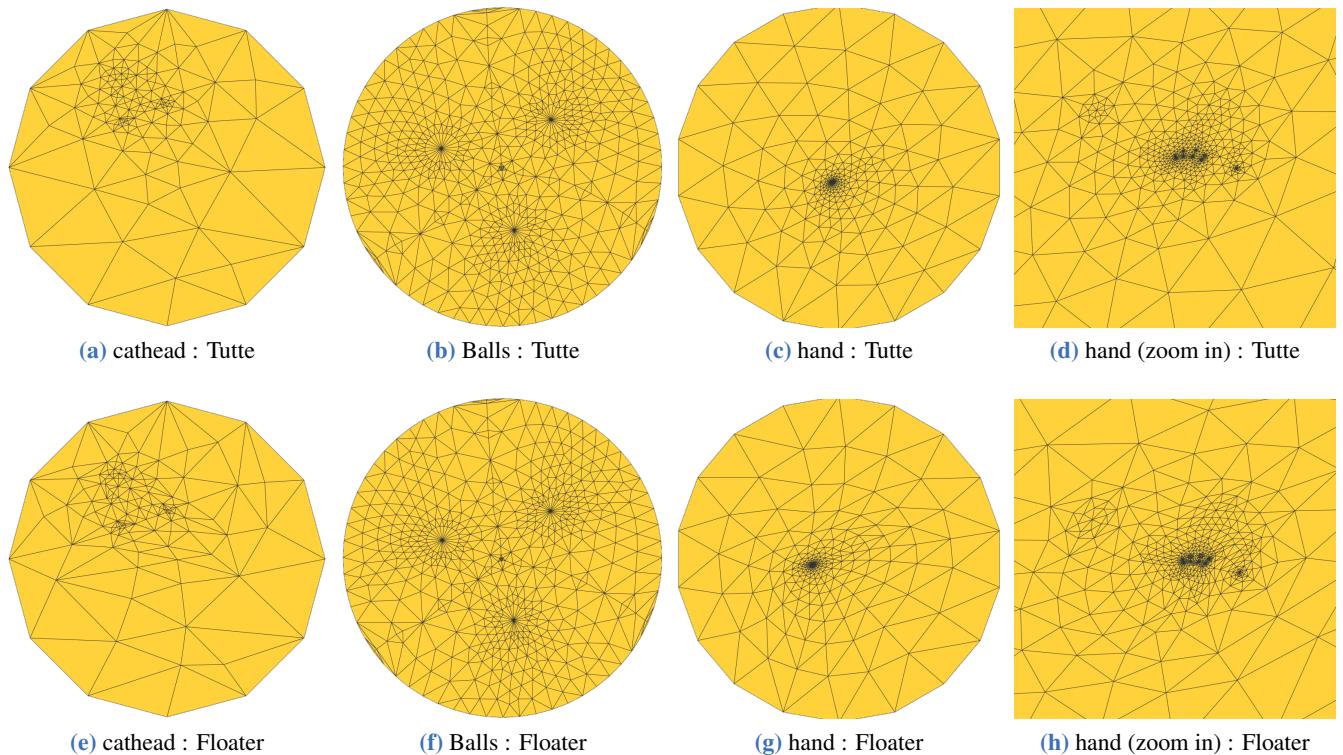


图 1.2: 测试数据渲染图 (边界形状: 正多边形)

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

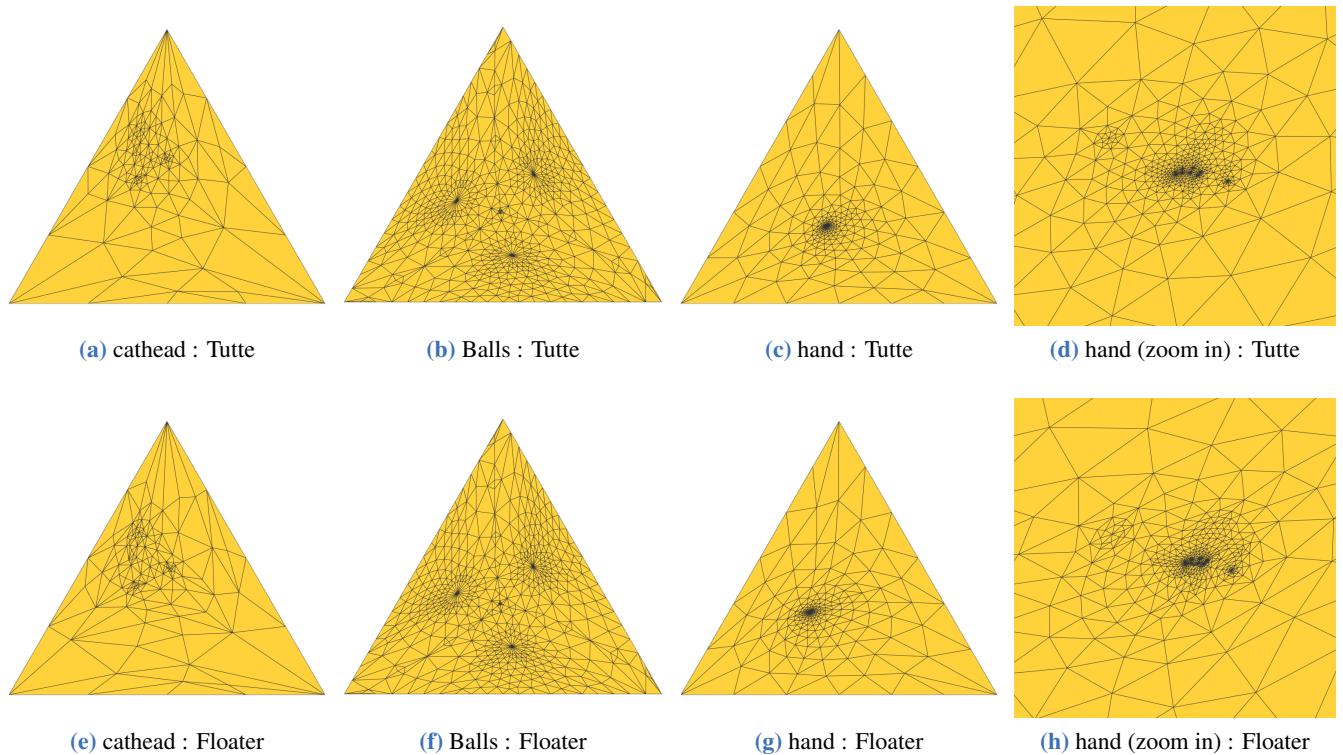


图 1.3: 测试数据渲染图 (边界形状: 正三角形)

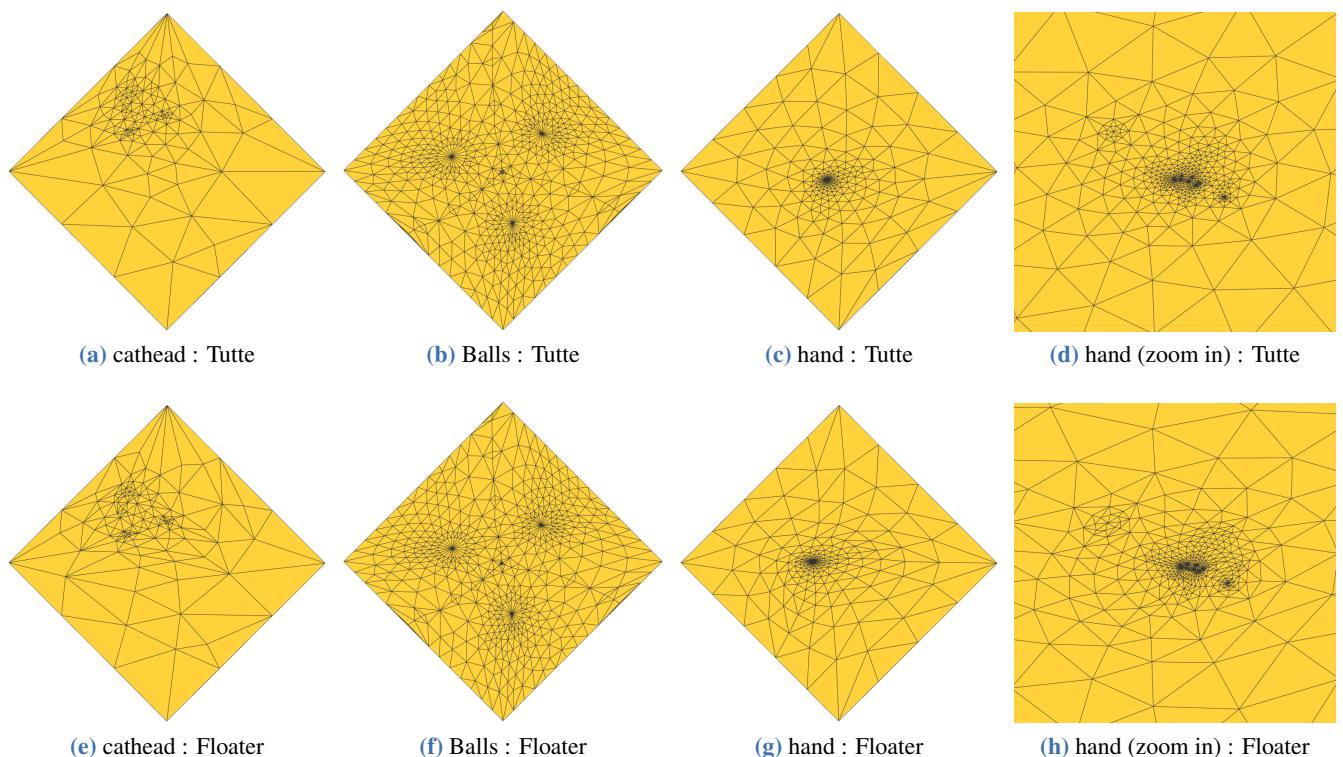


图 1.4: 测试数据渲染图 (边界形状: 正四边形)

大地扭曲，在下一节的纹理映射中可以观察到这种现象。

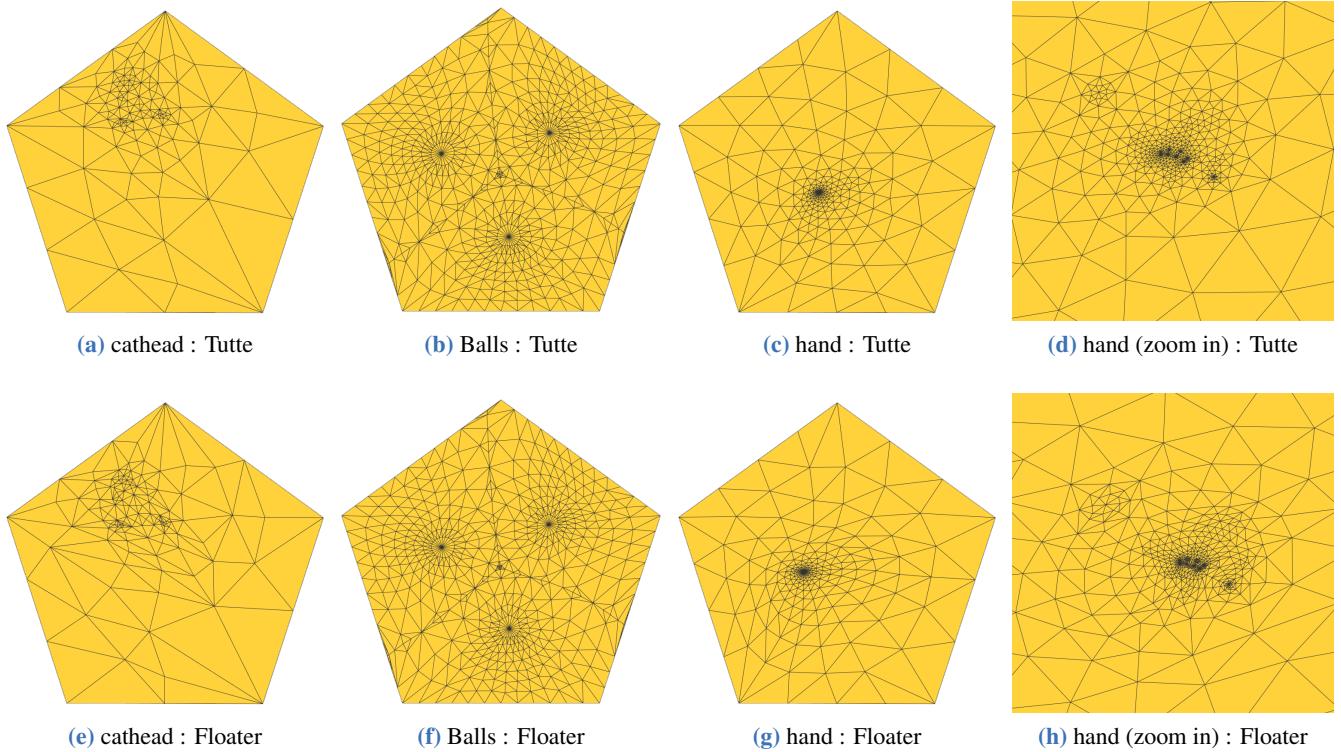


图 1.5: 测试数据渲染图 (边界形状: 正五边形)

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。

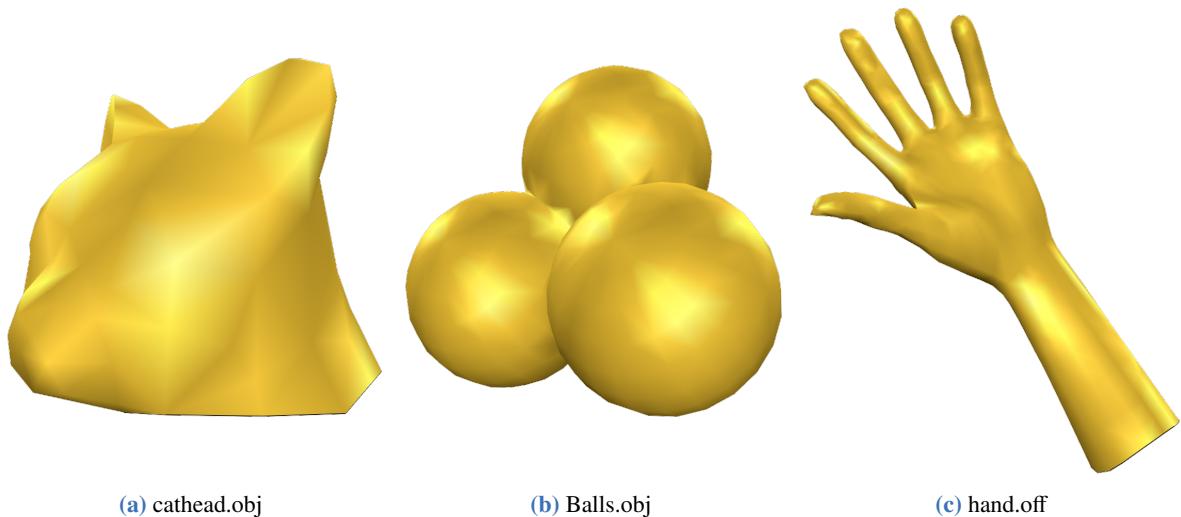


图 1.6: 测试数据渲染图 (平滑着色)

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。

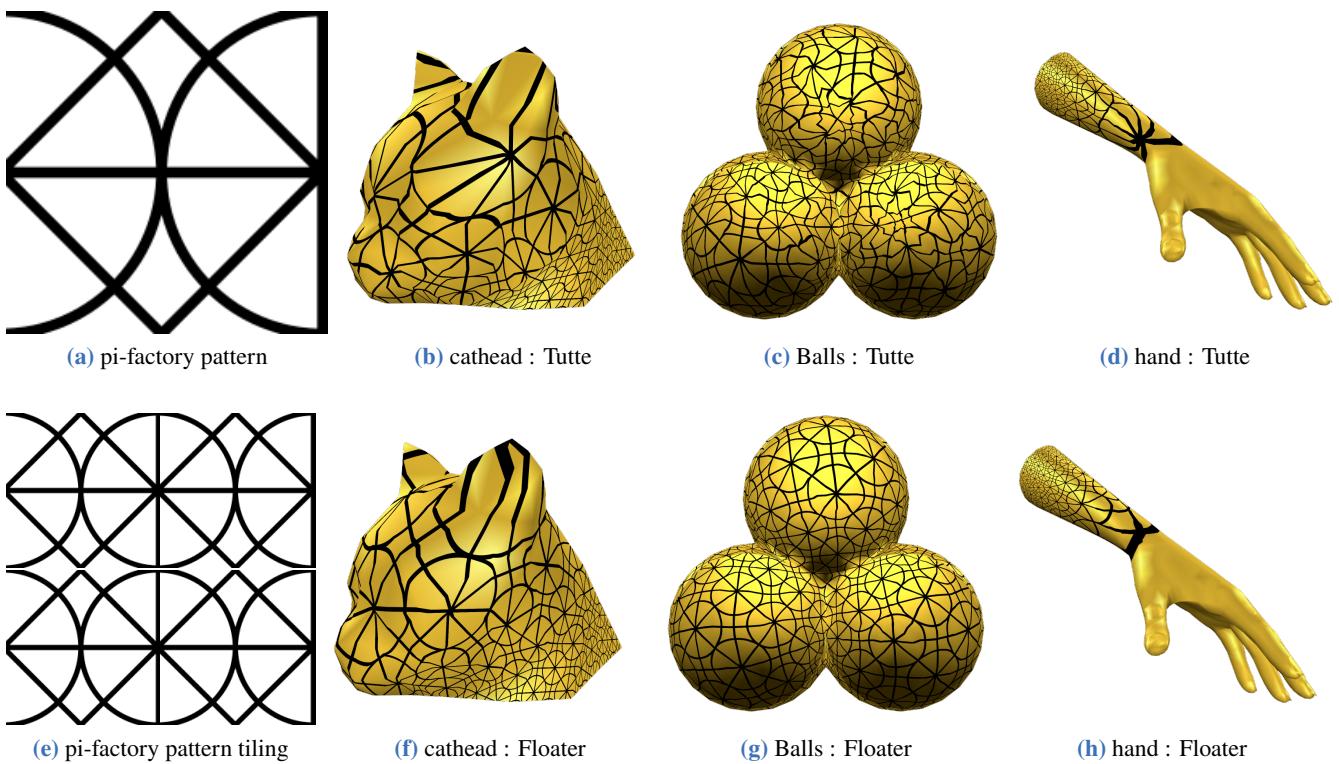


图 1.7: 测试数据纹理映射渲染图 (边界形状: 正多边形)

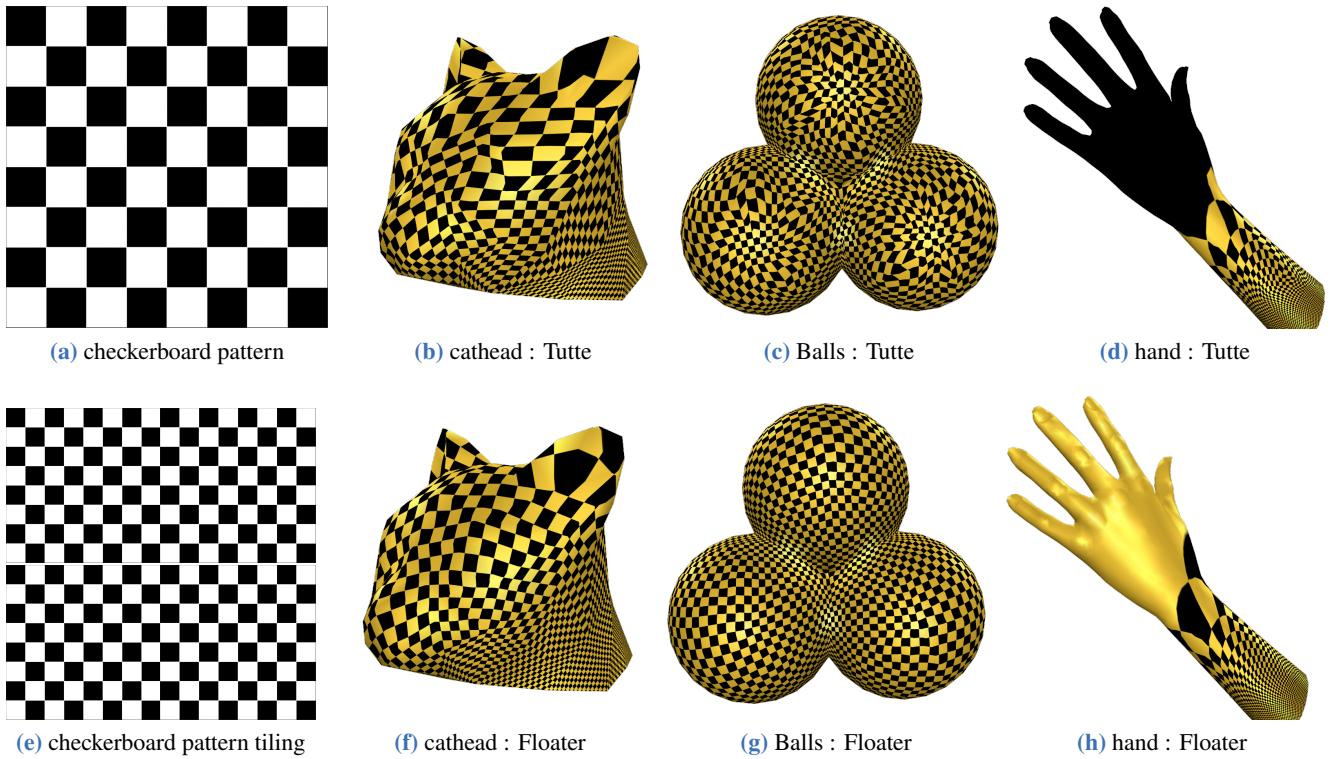


图 1.8: 测试数据纹理映射渲染图 (边界形状: 正多边形)

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

```
1 // ===== surfacemeshprocessing.h =====
2 // add QAction for smooth rendering
3 // ...
4 QAction *actSmooth;
5 // ...
6 // add QAction for parameterization
7 QAction *actParamAverage;
8 QAction *actParamFloater;
9 // ...
10
11 // ===== surfacemeshprocessing.cpp =====
12 void SurfaceMeshProcessing::CreateActions(void)
13 {
14     // ...
15     // smooth shading
16     actSmooth = new QAction(tr("Smooth"), this);
17     actSmooth->setIcon(QIcon(":/SurfaceMeshProcessing/Images/smooth.png"));
18     actSmooth->setStatusTip(tr("Show smooth"));
19     actSmooth->setCheckable(true);
20     connect(actSmooth, SIGNAL(triggered()), viewer, SLOT>ShowSmooth()));
21     // ...
22     agViewGroup->addAction(actSmooth);
23     // ...
24     // parameterization action
25     actParamAverage = new QAction("Average Weighted", this);
26     actParamAverage->setStatusTip(tr("Neighbors Weighted Averagely"));
27     connect(actParamAverage, SIGNAL(triggered()), viewer, SLOT>TutteParam_AverageWeight
28     ());
29
30     actParamFloater = new QAction("Floater Weighted", this);
31     actParamFloater->setStatusTip(tr("Neighbors Weighted Floaterly"));
32     connect(actParamFloater, SIGNAL(triggered()), viewer, SLOT>TutteParam_FlaterWeight
33     ());
34
35 void SurfaceMeshProcessing::CreateMenus(void)
36 {
37     // ...
38     QMenu* menuTools = menuBar()->addMenu(tr("&Tool"));
39     QMenu* tutteParam = menuTools->addMenu(tr("Tutte's Param"));
```

```

40     tutteParam->addAction(actParamAverage);
41     tutteParam->addAction(actParamFloater);
42     // ...
43 }
44
45 void SurfaceMeshProcessing::CreateToolBars(void)
46 {
47     // ...
48     tbView->addAction(actSmooth);
49     // ...
50 }
```

Listing A.2: MainViewerWidget

```

1 // ===== MainViewerWidget.h =====
2 class MainViewerWidget : public QDialog
3 {
4     // ...
5     // Tutte's parameterization API
6     void TutteParam_AverageWeight();
7     void TutteParam_FlaterWeight();
8     // ...
9 }
10
11 // ===== MainViewerWidget.cpp =====
12 void MainViewerWidget::TutteParam_AverageWeight()
13 {
14     meshviewerwidget->TutteParam(MeshViewerWidget::TutteParamType::AVERAGE_WEIGHTED);
15 }
16
17 void MainViewerWidget::TutteParam_FlaterWeight()
18 {
19     meshviewerwidget->TutteParam(MeshViewerWidget::TutteParamType::FLOATER_WEIGHTED);
20 }
```

Listing A.3: MeshViewerWidget

```

1 // ===== MeshViewerWidget.h =====
2 class MeshViewerWidget : public QGLViewerWidget
3 {
4     // ...
5     // parameterization enums
6     enum class TutteParamType { AVERAGE_WEIGHTED, FLOATER_WEIGHTED };
7     // parameterization functions
8     void TutteParam(TutteParamType type);
9     // ...
10    void DrawSmooth() const;
11    // ...
12 }
```

```

13
14 // ===== MeshViewerWidget.cpp =====
15
16 // smooth shading mode
17 void MeshViewerWidget::DrawSmooth() const
18 {
19     glShadeModel(GL_SMOOTH);
20
21     glBindTexture(GL_TEXTURE_2D, glTextureID);
22     glEnable(GL_TEXTURE_2D);
23
24     glBegin(GL_TRIANGLES);
25
26     for (const auto& fh : polyMesh->polyfaces())
27     {
28         for (const auto& fvh : polyMesh->polygonVertices(fh))
29         {
30             glNormal3dv(fvh->normal().data());
31             auto uv = fvh->getTextureUVW().uv;
32             float uvScale = 10.0f;
33             glTexCoord2f(uv[0] * uvScale, uv[1] * uvScale);
34             glVertex3dv(fvh->position().data());
35         }
36     }
37
38     glEnd();
39 }
40
41 //===== my parameterization functions =====
42 enum class UVBoundaryType {
43     POLYGON_CIRCLE,
44     POLYGON_TRIANGLE,
45     POLYGON_SQUARE,
46     POLYGON_PENTAGON,
47     // POLYGON_STAR,
48     // POLYGON_CROSS
49 } ;
50
51 std::vector<glm::dvec2> GetBoundaryUVs(size_t numVerts, UVBoundaryType type)
52 {
53     const double CIRCLE_RADIUS = 0.5;
54     std::vector<glm::dvec2> boundaryUVs(numVerts, { CIRCLE_RADIUS, CIRCLE_RADIUS });
55
56     // 1. polygon edges
57     size_t polygonEdges = 3;
58     switch (type)
59     {
60         case UVBoundaryType::POLYGON_CIRCLE:    polygonEdges = numVerts;      break;
61         case UVBoundaryType::POLYGON_TRIANGLE:   polygonEdges = 3;           break;

```

```

62     case UVBoundaryType :: POLYGON_SQUARE:    polygonEdges = 4;           break;
63     case UVBoundaryType :: POLYGON_PENTAGON:   polygonEdges = 5;           break;
64   }
65
66   polygonEdges = std::min(polygonEdges, numVerts);
67
68 // 2. initialization
69 std::vector<int> polygonEdgePoints(polygonEdges, 0);
70 {
71     int V = numVerts;
72     int i = 0;
73     while (V-- > 0)
74     {
75         polygonEdgePoints[i]++;
76         i = (i + 1) % polygonEdges;
77     }
78 }
79 assert(std::accumulate(polygonEdgePoints.begin(), polygonEdgePoints.end(), 0) ==
80 numVerts);
81
82 double angleGap = 2.0 * M_PI / static_cast<double>(polygonEdges);
83 double edgeLength = 2.0 * CIRCLE_RADIUS * glm::sin(angleGap / 2.0);
84
85 // rotate matrix
86 glm::dmat2x2 polygonRotateMat = glm::dmat2({ 0, 1 }, { -1, 0 });
87
88 // 3. calculation
89 auto boundaryUVIter = boundaryUVs.begin();
90 for (size_t edgeIdx = 0; edgeIdx < polygonEdges; ++edgeIdx)
91 {
92     glm::dvec2 basePoint = CIRCLE_RADIUS * glm::dvec2{ glm::cos(edgeIdx * angleGap),
93 , glm::sin(edgeIdx * angleGap) };
94     glm::dvec2 edgeDirection = CIRCLE_RADIUS * glm::dvec2(glm::cos(((edgeIdx + 1) %
95 polygonEdges) * angleGap),
96                                         glm::sin(((edgeIdx + 1) %
97 polygonEdges) * angleGap)) - basePoint;
98
99     int edgePoints = polygonEdgePoints[edgeIdx];
100    double edgeDisDelta = edgeLength / static_cast<double>(edgePoints);
101
102    int i = 0;
103    while (i++ < edgePoints) {
104        glm::dvec2 pointUV = basePoint + (i / static_cast<double>(edgePoints)) *
105 edgeDirection;
106        *boundaryUVIter += (polygonRotateMat * pointUV);
107        boundaryUVIter++;
108    }
109 }
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
16
```

```

106     return boundaryUVs;
107 }
108
109 // return sum theta and { ||xj - xi||, theta_j } of each xj respectively
110 auto FloaterParam_I_a(acamcad::polymesh::MVert* v,
111     const std::vector<acamcad::polymesh::MVert*>& adjVerts)
112 {
113     double thetaI{ 0.0 };
114     int N = adjVerts.size();
115
116     std::vector<glm::dvec2> disANDangles(N, { 0.0, 0.0 });
117
118     for (int i = 0; i < N; ++i)
119     {
120         int j = (i + 1) % N;
121         auto vecI = adjVerts[i]->position() - v->position();
122         auto vecJ = adjVerts[j]->position() - v->position();
123
124         glm::dvec3 vi(vecI.x(), vecI.y(), vecI.z());
125         glm::dvec3 vj(vecJ.x(), vecJ.y(), vecJ.z());
126
127         double di = glm::sqrt(glm::dot(vi, vi));
128         double dj = glm::sqrt(glm::dot(vj, vj));
129
130         disANDangles[i].x = di;
131         disANDangles[i].y = glm::acos(glm::dot(vi, vj) / (di * dj));
132         thetaI += disANDangles[i].y;
133     }
134     assert(thetaI > 0);
135     return std::make_tuple(thetaI, disANDangles);
136 }
137
138 // return UVs of neighbor vertices of v
139 auto FloaterParam_I_b(double thetaSum, const std::vector<glm::dvec2>& neighborAngleInfo
140 )
141 {
142     int N = neighborAngleInfo.size();
143
144     glm::dvec2 P(0.0, 0.0);
145     std::vector<glm::dvec2> adjUVs(N, { 0.0, 0.0 });
146     adjUVs[0] = { neighborAngleInfo[0].x, 0.0 };
147
148     for (int j = 1; j < N; ++j)
149     {
150         // 1. rotate factor
151         double thetaI = neighborAngleInfo[j - 1].y * (2.0 * M_PI / thetaSum);
152         double cosThetaI = glm::cos(thetaI);
153         double sinThetaI = glm::sin(thetaI);
154         glm::dmat2x2 rotateMat2({ cosThetaI, sinThetaI }, { -sinThetaI, cosThetaI });

```

```

154
155     // 2. scale factor
156     double scale = neighborAngleInfo[j].x / neighborAngleInfo[j - 1].x;
157     double dP = scale * glm::sqrt(glm::dot(adjUVs[j - 1] - P, adjUVs[j - 1] - P));
158     glm::dmat2x2 scaleMat2({ scale, 0.0 }, { 0.0, scale });
159
160     adjUVs[j] = scaleMat2 * rotateMat2 * adjUVs[j - 1];
161 }
162
163 return adjUVs;
164 }
165
166 // If triangle P1P2P3 centering the original point (0, 0)
167 bool IsInTriangle(glm::dvec2 P1, glm::dvec2 P2, glm::dvec2 P3)
168 {
169     double t1 = P1.x * P2.y - P1.y * P2.x;
170     double t2 = P2.x * P3.y - P2.y * P3.x;
171     double t3 = P3.x * P1.y - P3.y * P1.x;
172
173     return t1 * t2 >= 0 && t1 * t3 >= 0 && t2 * t3 >= 0;
174 }
175
176 auto FloaterParam_II(const std::vector<glm::dvec2>& adjUVs)
177 {
178     int N = adjUVs.size();
179     std::vector<double> mu_ls(N, 0.0);
180
181     for (int i = 0; i < N; ++i) {
182
183         auto Pl = adjUVs.begin() + i;
184
185         auto Pr = adjUVs.begin() + (i + 1) % N;
186         auto Pn = adjUVs.begin() + (i + 2) % N;
187
188         auto IterAscend = [&]() {
189             Pr++;
190             Pn++;
191             if (Pr == adjUVs.end())    Pr = adjUVs.begin();
192             if (Pn == adjUVs.end())  Pn = adjUVs.begin();
193         };
194
195         while (Pn != Pl)
196         {
197             if (!IsInTriangle(*Pl, *Pr, *Pn))
198             {
199                 IterAscend();
200                 continue;
201             };
202         }
203     }
204 }
```

```

203     Eigen::Matrix3d A;
204     A(0, 0) = Pl->x;
205     A(1, 0) = Pl->y;
206     A(2, 0) = 1.0;
207     A(0, 1) = Pr->x;
208     A(1, 1) = Pr->y;
209     A(2, 1) = 1.0;
210     A(0, 2) = Pn->x;
211     A(1, 2) = Pn->y;
212     A(2, 2) = 1.0;
213     Eigen::Vector3d b(0.0, 0.0, 1.0);
214
215     auto x = A.lu().solve(b);
216
217     mu_ls[Pl - adjUVs.begin()] += x(0);
218     mu_ls[Pr - adjUVs.begin()] += x(1);
219     mu_ls[Pn - adjUVs.begin()] += x(2);
220     mu_ls[i] += x(0, 0);
221
222     assert(!std::isnan(x(0) * x(1) * x(2)));
223
224     IterAscend();
225 }
226
227 std::for_each(mu_ls.begin(), mu_ls.end(), [N](double& v) { return v / static_cast<
228 double>(N); });
229 return mu_ls;
230 }
231
232 void AverageParam(acamcad::polymesh::MVert* v,
233 const std::vector<acamcad::polymesh::MVert*>& adjVerts,
234 std::vector<double>& weights)
235 {
236     weights.clear();
237     weights.resize(adjVerts.size(), 1.0);
238 }
239
240 void FloaterParam(acamcad::polymesh::MVert* v,
241 const std::vector<acamcad::polymesh::MVert*>& adjVerts,
242 std::vector<double>& weights)
243 {
244     int N = adjVerts.size();
245     weights.clear();
246
247     // Stage 1 : Initialize (u, v) list
248     // a) thetaI
249     auto [thetaI, disANDangles] = FloaterParam_I_a(v, adjVerts);
250 }
```

```

251 // b) UVs
252 auto UVs = FloaterParam_I_b(thetaI, disANDangles);
253
254 //== Stage 2 : Calculate lambda_[i, jk] ==
255 weights = FloaterParam_II(UVs);
256 }
257
258 std::vector<double> CalAdjectWeight(acamcad::polymesh::MVert* v,
259 const std::vector<acamcad::polymesh::MVert*>& adjVerts,
260 MeshViewerWidget::TutteParamType type)
261 {
262     std::vector<double> weights;
263
264     switch (type)
265     {
266         case MeshViewerWidget::TutteParamType::AVERAGE_WEIGHTED:
267             AverageParam(v, adjVerts, weights);
268             break;
269         case MeshViewerWidget::TutteParamType::FLOATER_WEIGHTED:
270             FloaterParam(v, adjVerts, weights);
271             break;
272     }
273     return weights;
274 }
275
276 void MeshViewerWidget::TutteParam(TutteParamType type)
277 {
278     using mat = Eigen::MatrixXd;
279     using acamcad::polymesh::MVert;
280     std::cout << "Tutte's Parameterization\n";
281
282     if (polyMesh->numVertices() == 0)
283     {
284         std::cerr << "ERROR: TutteParam() No vertices!" << std::endl;
285         return;
286     }
287
288     //== calculate boundary vertices ==
289     auto boundaryVertLists = polyMesh->boundaryVertices();
290     std::cout << "Boundary Points [" << boundaryVertLists.size() << "]\n";
291
292     // 1. prepare convex polygon
293     int M = boundaryVertLists.size();
294     int N = polyMesh->numVertices();
295     auto boundaryUVs = GetBoundaryUVs(M, UVBoundaryType::POLYGON_CIRCLE);
296     //auto boundaryUVs = GetBoundaryUVs(M, UVBoundaryType::POLYGON_TRIANGLE);
297     //auto boundaryUVs = GetBoundaryUVs(M, UVBoundaryType::POLYGON_SQUARE);
298     //auto boundaryUVs = GetBoundaryUVs(M, UVBoundaryType::POLYGON_PENTAGON);
299

```

```

300 // 2. prepare matrix (Eigen::Dense)
301 mat A = mat::Zero(N, N);
302 mat x = mat::Zero(N, 2);
303 mat b = mat::Zero(N, 2);
304
305 // a) boundary vertices
306 for (int i = 0; i < M; ++i)
307 {
308     int vertID = boundaryVertLists[i]->index();
309
310     A(vertID, vertID) = 1.0;
311     b(vertID, 0) = boundaryUVs[i].x;
312     b(vertID, 1) = boundaryUVs[i].y;
313 }
314
315 // b) inner vertices
316 std::unordered_set<MVert*> boundaryVertsDict(boundaryVertLists.begin(),
317 boundaryVertLists.end());
318
319 for (int i = 0; i < N; ++i)
320 {
321     auto pVert = polyMesh->vert(i);
322     if (boundaryVertsDict.count(pVert))    continue;
323
324     int vID = pVert->index();
325
326     auto adjVerts = polyMesh->vertAdjacentVertices(pVert);
327     auto weights = CalAdjectWeight(pVert, adjVerts, type);
328     double weightSUM = std::accumulate(weights.begin(), weights.end(), 0.0);
329     for (int j = 0; j < adjVerts.size(); ++j) {
330         auto adjID = adjVerts[j]->index();
331         A(vID, adjID) = weights[j];
332         A(vID, vID) = -weightSUM;
333     }
334 }
335
336 /// 3. solve the equation Ax = b
337 x = A.lu().solve(b);
338
339 std::cout << "Tutte's parameterization finished, updating mesh ... \n";
340
341 for (int i = 0; i < N; ++i)
342 {
343     auto pVert = polyMesh->vert(i);
344     auto vID = pVert->index();
345
346     //pVert->setPosition(x(vID, 0), x(vID, 1), 0.0);
347     pVert->setTexture(x(vID, 0), x(vID, 1));
348 }

```

```

348     std::cout << "Done\n";
349
350
351     UpdateMesh();
352     update();
353 }
```

Listing A.4: QGLViewerWidget

```

1 // ===== QGLViewerWidget.h =====
2 class QGLViewerWidget : public QOpenGLWidget
3 {
4     // ...
5     // load uv textures
6     void LoadTexture();
7 }
8
9 // ===== QGLViewerWidget.cpp =====
10 void QGLViewerWidget::LoadTexture()
11 {
12     glGenTextures(1, &glTextureID);
13     glBindTexture(GL_TEXTURE_2D, glTextureID);
14
15     // load and generate texture
16     int width, height, nChannels;
17     unsigned char* data = stbi_load("../src/Images/tiling patterns/R-C3.jpg", &width, &
18     height, &nChannels, 0);
19     //unsigned char* data = stbi_load("../src/Images/tiling patterns/circle.jpg", &
20     width, &height, &nChannels, 0);
21     //unsigned char* data = stbi_load("../src/Images/tiling patterns/pie-factory.png", &
22     width, &height, &nChannels, 0);
23     //unsigned char* data = stbi_load("../src/Images/tiling patterns/cross.jpg", &width
24     , &height, &nChannels, 0);
25
26     GLint internalFormat = GL_RGBA;
27     if (nChannels == 1) internalFormat = GL_RED;
28     if (nChannels == 2) internalFormat = GL_RG;
29     if (nChannels == 3) internalFormat = GL_RGB;
30     if (nChannels == 4) internalFormat = GL_RGBA;
31
32     if (data)    glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, width, height, 0,
33     internalFormat, GL_UNSIGNED_BYTE, data);
34     else
35     {
36         std::cout << "Failed to load texture" << std::endl;
37         assert(false);
38     }
39
40     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

```
36 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
37 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
38 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
39 glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
40 glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
41
42     stbi_image_free(data);
43 }
```

A.2 新增文件

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

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`

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.