arXiv:2512.19959v1 [cs.CG] 23 Dec 2025

边折叠网格简化综合指南

Purva Kulkarni
独立研究者

Aravind Shankara Narayanan
独立研究者

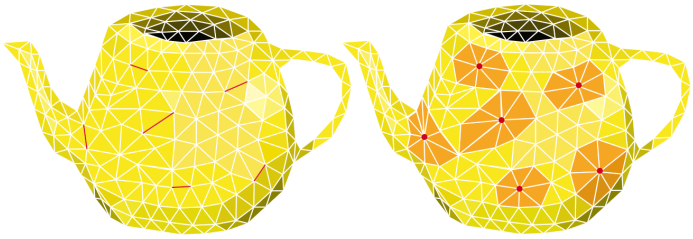

图 1。使用边折叠进行网格简化。

摘要

网格简化是指在保留网格整体形状及其显著特征的同时，减少三维（*3D*）网格中顶点、边和三角形数量的过程。一种广泛采用的策略是边折叠，即将连接两个顶点的一条边合并为单个顶点。待塌缩的边依据成本函数进行选择，该函数用于估算此次塌缩引入的误差。本文面向寻求理论基础与实践见解的从业者和研究人员，系统介绍并以实现为导向讲解边折叠。我们回顾并推导其基础数学原理，并为包括二次误差度量（*QEM*）和 *Lindstrom-Turk* 几何准则在内的基础代价函数提供参考实现。我们还解释 *QEM* 变体中的属性感知边折叠，以及渐进网格所采用的 *Hoppe* 能量法背后的数学原理。除代价函数外，我们还概述完整的边折叠算法，包括具体的操作顺序和通常使用的数据结构。为了构建稳健的系统，我们还介绍必要的程序保护措施，以防止网格退化、法线翻转以及边界条件处理不当等问题。本文的目标不仅是整合已有方法，还要弥合理论与实践之间的差距，为实现基于边折叠的网格简化流程提供清晰、循序渐进的指南。

1. 引言

三角形是计算机图形学中最常用的绘制图元。几乎所有图形库和硬件系统都原生支持三角形，因此三角形网格成为三维建模中的主流表示形式。得益于数十年的硬件发展，现代图形系统能够渲染由数百万个三角形组成的模型。然而，随着摩尔定律逐渐放缓，以及网格的几何复杂度迅速提升，单纯依赖暴力并行处理已不再可行。因此，网格简化技术对于交互式和大规模应用实现实时性能与可扩展性，比以往任何时候都更加重要。网格简化构成了细节层次（LOD）系统的基础，可减轻图形处理器的工作负载，加速游戏中的碰撞检测，并支持在有限元分析仿真中更快速地生成粗略近似。

在各种网格简化技术中，边折叠在实践中应用最广泛。许多主流图形库和工具都实现了这一策略，例如*CGAL* 、*QSlim*和*meshoptimizer*。边折叠操作会将一条边的两个端点合并为一个新顶点，从而移除该边以及与之相邻的两个三角形。反复执行这一操作，便能得到一个保留原始网格整体结构的简化网格。成本函数有助于确定要折叠哪条边，以及如何放置生成的顶点，从而尽可能保留模型的视觉和几何细节。

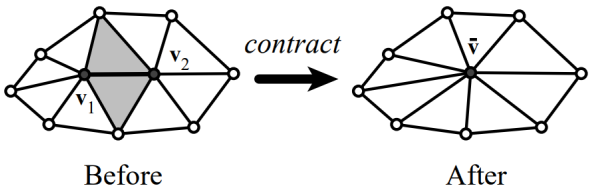

图 2。边折叠

尽管网格简化已得到广泛研究，但该领域的新手在阅读基础论文时往往会面临陡峭的学习曲线。许多这类论文侧重于最终方程或高层次的算法描述，却很少深入说明其背后的几何推理或实际实现细节。因此，读者可能难以直观理解基于边折叠的简化如何运作、为何有效，也难以将理论转化为可运行的代码。

本文旨在通过对流形网格上的基于边折叠的网格简化进行详尽且兼顾实现的分析，弥合这一差距。我们的贡献如下：

- 我们提出了一套完整的端到端简化流程，其中包括用于表示网格连通性的精心选择的数据结构、对该领域基础论文中所提出的代价函数的深入分析，以及将二者结合起来的边折叠算法。

- 与许多仅给出最终代价指标或优化函数的既有研究不同，我们推导并解释这些内容，同时阐明这些公式背后的几何意义，使读者能够理解每个步骤的原理。

- 我们的目标有两个：为希望理解简化算法内部运作机制的学习者提供概念指南，并为希望实现自有系统的开发者提供实用参考。

本文首先对不同类别的网格简化算法进行分类和回顾。由于边折叠操作的效率取决于能否快速访问网格连通性并迅速进行局部更新，接下来一节将介绍可用于存储和管理网格连通性信息的数据结构。随后，我们将介绍一种全面的边折叠算法，其中包括用于防止网格退化的详细程序检查。篇幅最长的一节将探讨代价计算策略，并结合实际实现，阐释基础论文中的数学公式。下一节将介绍考虑逐顶点属性的高级边塌缩技术。最后，我们将提供支持这些技术的补充数学结果和证明。

2. 网格简化技术的类别

网格简化技术多种多样，但大多数都可以根据其降低几何复杂度、同时保持拓扑的策略进行分类，如 [Cignoni 等人 1998]所述。我们在此基础上补充了该领域的最新进展，这些进展借助了机器学习和神经网络等现代技术。

早期的网格减面策略着重于检测共面或近乎共面的曲面片，并将它们合并成更大的多边形区域，如 [De-Haemer Jr 和 Zyda 1991]以及 [Hinker 和 Hansen 1993]所述。随后，这些区域会经过重新三角剖分，生成面数更少的网格。尽管方法简单，却常常会损失几何细节，并引入拓扑不一致。

另一种称为顶点聚类的方法，会根据空间距离将彼此靠近的顶点分组，并用一个代表顶点替换每个聚类，随后进行局部重新三角剖分，如 [Rossignac 和 Borrel 1993] 所述，并由 [Low 和 Tan 1997]加以改进。虽然速度更快，但研究发现，这种方法同样会损害细节和拓扑准确性。

一种更精细且注重拓扑的方法是迭代局部减面，它基于局部几何评估，逐步移除顶点、边或面。

这些操作通常由成本函数引导，旨在保留网格的整体结构和外观 [Garland and Heckbert1997；Lindstrom and Turk 1998；Schroeder 等人 1992]。简化包络等扩展 [Cohen 等人 1996]通过强制生成的简化网格位于两个偏移网格之间，实现了有界误差控制。

在基于能量的优化方法中，例如 [Hoppe 等人提出的方法，1993], 全局成本函数会评估网格的整体质量。简化通过迭代执行基于边的操作来完成，例如塌缩、交换或分裂，其目标是最小化局部和全局成本函数。尽管这种采用全局优化的方法有望更好地保留整体结构，但由于计算复杂度较高，在实践中较少使用。

另一种策略是重新铺网，由 [Turk 1992], 提出。该方法首先在原始曲面上随机放置用户指定数量的较少新顶点，然后根据曲率较高的区域调整这些顶点，并基于这一顶点集合构建新的简化三角剖分。尽管这种方法能有效减少三角形数量，但它不支持逐顶点属性，因此不太适用于计算机辅助设计（CAD）或物理仿真等必须使用此类数据的应用。

另一种值得关注的网格简化方法是体素化，He 等人 [提出的工作 1995] 以及 [He 等人 1996]的工作都采用了这种方法。在这种方法中，首先将网格采样到体素网格中，再在每个网格点应用低通滤波器，生成离散标量场。随后，根据滤波器确定的等值，使用标准移动立方体算法或其自适应变体从该标量场中提取三角网格曲面。基于体素的网格的细节程度可通过分辨率进行调整，但这种方法在工业中的应用有限。它会平滑尖锐特征，因此不适用于 CAD；同时，体处理计算开销大，且缺乏明确的几何误差控制，使得输出质量难以保证。

近期研究探索了神经网络方法，这些方法要么直接简化网格，要么提供支持细节层次控制的隐式表示。 [Potamias 等人 2022]采用可微神经网络，通过稀疏注意力机制选择输入顶点的一个子集，并对选出的顶点重新三角剖分，从而以数据驱动且具备泛化能力的方式生成简化网格，无需逐网格重新训练。 [Chen 等人 2023] 使用二次误差度量生成粗略基础网格，随后通过面分裂进行神经网络重新网格化。逐面潜在特征表示会传输到客户端并在那里解码，以重建更精细的网格。该方法能够隐式生成跨多个细节层次的简化表示。 [Park 等人 2019] 从网格的体素化表示中学习有符号距离场（SDF）表示。 [Takikawa 等人 2021] 通过创建多尺度有符号距离场对其进行扩展，使其能够通过光线步进在不同细节层次下实现实时渲染。尽管可以使用移动立方体等方法提取简化三角网格，但这会削弱其隐式表示的效率。

3. 表示网格连通性的数据结构

网格连通性数据结构旨在高效组织和管理网格元素之间的关系，例如哪些面共用一条边、哪些边与某个顶点相连，或哪些顶点构成一个面。它们使算法能够快速遍历和操作网格的拓扑。下面介绍两种常用于表示网格连通性的数据结构，并评估它们对边折叠操作的支持能力。

CornerTable 数据结构由 [Rossignac 2002] 提出，是一种紧凑的网格表示方式，其中每个三角形的三个“角”（顶点-三角形关联）都存储在一个列表中。对于边折叠，它能高效管理边折叠更新，并支持快速查询网格。

半边 数据结构由 [McGuire 2000], 提出，因其直观的设计以及在各类网格库中的广泛支持而得到广泛应用。在这种结构中，每条网格边都由一对方向相反的半边表示，每条半边都存储与相关元素（如顶点、面和相邻边）的连通信息。虽然它并非内存效率最高的方案，但能够实现快速网格查询和局部更新，因此非常适用于边折叠等操作。

在下面的代码清单中，我们将介绍典型的连通性数据结构所支持的接口。

```
类 IVertex
    { virtual vec3 获取位置() = 0; virtual vecn GetAttributes() = 0; }; class IEdge {
virtual vector<IVertex*> 获取顶点() = 0; }; class IFace { virtual vector<IVertex*>
获取顶点() = 0; virtual vector<IEdge*> GetEdges() = 0; }; class IMesh { virtual
vector<Face*>GetConnectedFaces(Edge* edge) = 0; virtual vector<Face*>
GetConnectedFaces(Vertex*vertex) = 0; virtual vector<Edge*>
GetConnectedEdges(Vertex*vertex) = 0; virtual vector<Vertex*>获取连通顶点
(Vertex* vertex) = 0; };
```

表中列出的查询1 对于基于边折叠的网格简化算法是必需的，因此所选的网格连通性数据结构必须高效地处理这些查询。正如表1所示，Half-Edge 和 Corner Table 结构都能够以最优的时间复杂度处理这些查询，因此非常适合基于边折叠的网格简化。

网格查询	**时间复杂度** （半边 / Corner Table）
获取与顶点 v 相连的所有三角形	O(度数(v))
获取与顶点 v 相连的所有边	O(度数(v))
获取与顶点 v相连的所有顶点	O(度数(v))
获取与顶点 v相连的所有边	O(度数(v))

表1。使用不同数据结构时的网格查询操作及其时间复杂度

4. 基于边折叠的网格简化算法

基于边折叠的简化算法会迭代减少网格中的三角形数量，同时保留其整体形状和特征。不同实现中的核心算法大体一致，主要区别在于代价度量和顶点放置策略。该算法通常包含以下步骤:

代价分配和最优顶点放置计算

1. 为网格中的每条边计算代价，以估算塌缩该边引入的几何误差。同时，确定合并后顶点的最优位置。这一步至关重要，因为大多数基于边折叠的简化策略都在此处有所不同。

2. 将计算出的代价、对应的边及其最优替代顶点存入优先队列。

迭代边折叠

目标三角形数量由程序内部定义，或由客户端代码在外部指定。随后，重复以下步骤，直到达到目标三角形数量:

1. 从优先队列中选取塌缩代价最低的边。

2. 执行有效性检查，确保塌缩该边后仍能保持网格的流形性。（我们采用的三项有效性检查将在下文说明。）

3. 如果该边通过所有有效性检查，就用计算出的顶点替换这条边，并移除相邻的两个三角形，从而完成塌缩。

4. 由于塌缩会在局部改变网格，因此重新计算与折叠边相连的所有边的代价，并更新优先队列中的相应条目，以确保下一次迭代的准确性。

5. 更新网格的连通性数据结构，以反映塌缩所做的更改。

边折叠有效性检查

检查 1 和检查 2 遵循以下文献确立的标准： [Hoppe 等人 1993], , 而检查 3 则是根据经验得出的。这些检查对于避免可能导致无效或非流形网格结构的退化至关重要。

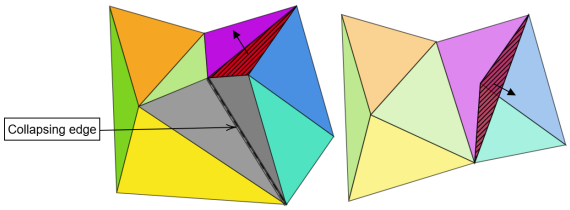

图 3。翻转的红色三角形，导致局部网格退化

1. **三角形翻转检查**：确保塌缩该边不会使任何周围三角形的方向发生求逆。塌缩边导致此类求逆的情况如图3所示。

```
bool面是否翻转(const IFace* 旧_面,const IFace*新_面){返回 点积
(util::ComputeNormal(旧_面), util::ComputeNormal(新_面)) < 0.
0;}
```

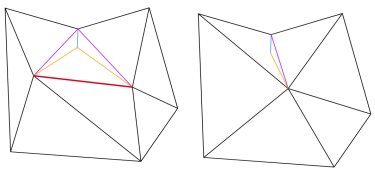

图4。塌缩红色边导致的非流形三角形形成

2. **双邻接顶点连通性检查**：验证折叠边两侧是否各有且仅有一对边合并。当折叠边的两个顶点恰好共享两个公共邻接顶点时，此条件成立。图4展示了与连通性相关的非流形三角形形成情况图4。

```
布尔值HasMoreThanTwoNeighbors(常量IMesh* 网格,常量IEdge*
    塌缩_边)
{
  auto 顶点 = 塌缩_边->获取顶点();
  auto conn_verts_v0 = 网格->获取连通顶点(顶点[0]);
  auto conn_verts_v1 = 网格->获取连通顶点(顶点[1]);

  集合<IVertex*> conn_verts_v1_集合(conn_verts_v1.begin(),
      连接_顶点_v1。结束());

  整数 公共_计数 = 0;
  对于(自动 顶点： 连接_顶点_v0)
    如果(连接_顶点_v1_集合。包含(顶点))
    {
      通用_计数++;
      如果 (共同_计数 > 2) 则返回 true;
    }
  返回 false;
}
```

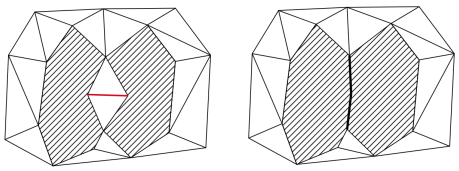

图5。边折叠合并内部边界，形成结构不良的局部网格

3. **边界合并检查**：如果一个面的除折叠边之外的另外两条边都位于网格边界或孔洞上，那么塌缩该边可能会导致两个边界合并，从而造成结构不良，如图5所示。

```
bool HasMultipleConnectedBoundaries(const IMesh* mesh, const IEdge*
collapse_边) { for(auto face : mesh->GetConnectedFaces(collapse_边)) { int
n_边_on_边界 = 0; for(auto edge : face->GetEdges()) { if(edge == 塌缩_边)
continue;
```

```
        如果(util::IsBoundaryEdge(边)) n_边_位于_边界++;
      }
      如果(n_边_位于_边界 > 1) 返回 true;
    }
    返回 false;
}
```

5. 代价函数

在基于边折叠的网格简化中，会为每条边分配一个误差度量，用于估计折叠该边的代价。误差最低的边会优先进行折叠。此外，我们还需要有效的策略来确定最佳新顶点位置，以替代折叠边，同时将几何失真降至最低。

以下 IConstraint类定义了误差度量的接口。实现此接口的成本函- 数类会计算候选顶点位置 $\mathcal{E}(v)$ 处折叠一条边的代价 v。此类的实现会将代价计算为 $\mathcal{E}(v) = v^T Hv + 2c^T v + k$，并将该方程中的实体 $\{H, c, k\}$ 存储在 m_H、m_c、m_k中。这些实体也将用于求得最优顶点放置，详情见 第 7 节。

```
类 IConstraint
{
  公有:
    虚函数 void评估代价(常量 三维向量 &v) = 0;

    常量 三阶矩阵&获取 H() 常量 { 返回 m_H; }
    常量 三维向量&获取 C() 常量 { 返回 m_c; }

  私有:
    三阶矩阵 m_H;
    三维向量 m_c;
    双精度 m_k;
}
```

5.1. 基于平面的二次型误差度量（QEM）

Garland 和 Heckbert 在 [中描述了这种方法 1997]，，该方法将误差定义为新顶点到周围三角形平面的距离之和，并将每个顶点视为这些平面的交集。这反映了新顶点偏离原始几何体的程度，以及由此产生的失真。

参见图 6，我们将边 (v_1, v_2) 塌缩为一个新顶点 v，移除两个相邻的三角形（分别位于平面 P_1 和 P_6中），并形成局部几何近似。误差通过计算 v 到原始周围平面P_1（经过 P_{10}）的距离之和来衡量。

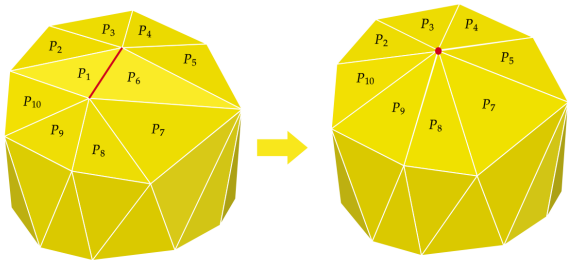

图 6　．QEM 边折叠周围的局部结构

令 $\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$ 为包围待塌缩边的平面集合。令 $\mathcal{E}$ 为新添加的顶点 v 引入的误差，其定义如下：

$$\mathcal{E} = \sum_{(n,d)\in\mathbb{P}} \left(n^T v + d\right)^2$$

其中，(n, d) 表示每个平面 $P \in \mathbb{P}$ 方程 $n \cdot r + d = 0$ 中的单位法向量 n 和标量 d。

项 $n^T v + d$ 表示顶点到平面的有符号距离。对其平方可确保误差始终非负，并对正向和负向偏差给予相同惩罚。

展开后，

$$\begin{aligned}
\mathcal{E} &= \sum_{(n,d)\in\mathbb{P}} \left(v^T n + d\right)\left(n^T v + d\right) \\
&= \sum_{(n,d)\in\mathbb{P}} \left(v^T \left(nn^T\right) v + 2dn^T v + d^2\right) \\
&= v^T\left(\textstyle\sum_{(n,d)\in\mathbb{P}} nn^T\right)v + 2\left(\sum_{(n,d)\in\mathbb{P}} dn\right)^T v + \textstyle\sum_{(n,d)\in\mathbb{P}} d^2 \\
&= v^T H v + 2c^T v + k
\end{aligned}$$

为了使误差最小，我们令其梯度为零，并求解 v:

$$\begin{aligned}
\nabla\mathcal{E} = 2Hv + 2c = 0 \\
\Longrightarrow v = -H^{-1}c
\end{aligned}$$

注意：当 $\mathcal{E}$ 取这种形式时，矩阵 H 就是它的海森矩阵。在这个特定情况下，H是半正定的。因此，使 $\nabla\mathcal{E} = 0$ 成立的点对应的是极小点，而非极大值点或鞍点。

如果矩阵 H 不可逆（即 $\det(H) = 0$），就无法通过这种方式计算最优顶点位置。在这种情况下，会采用回退策略或替代约束。对于这个成本函数，不可逆的 H 表示边折叠周围的曲面是平坦的，如下所述：

$$
\begin{aligned}
H &= \sum nn^T \qquad \text{where } n = \begin{pmatrix} a & b & c \end{pmatrix}^T \\
&= \sum \begin{pmatrix} a \\ b \\ c \end{pmatrix} \begin{pmatrix} a & b & c \end{pmatrix} \\
&= \sum \begin{pmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{pmatrix}
\end{aligned}
$$

请注意，$\begin{pmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{pmatrix} = \begin{pmatrix} an & bn & cn \end{pmatrix}$ 采用分块表示法。这意味着求和中的每一项本身都是不可逆矩阵，因为它的所有列都平行于 n。因此，当 H 不可逆时，构成 H 的所有平面的法线都彼此平行。这种情况会在局部曲面平坦时出现。

注意：此方法名称中的“二次型”一词，源于当 v 以四维齐次坐标表示为向量 $\begin{pmatrix} v \\ 1 \end{pmatrix}$时，这种误差所呈现的形式。在这种情况下，误差 ε 表示如下：

$$
\varepsilon = \begin{pmatrix} v & 1 \end{pmatrix}^T \begin{pmatrix} H & c \\ c^T & k \end{pmatrix} \begin{pmatrix} v \\ 1 \end{pmatrix}
$$

其中，作者将上文的 4x4 矩阵 $Q = \begin{pmatrix} H & c \\ c^T & k \end{pmatrix}$ 定义为这条边的总误差二次型。它还可以分解为每个平面 $P \in \mathbb{P}$对应的基本误差二次型 K_P之和：

$$
Q = \begin{pmatrix} H & c \\ c^T & k \end{pmatrix} = \sum_{P \in \mathbb{P}} K_P, \; K_P = \begin{pmatrix} nn^T & dn^T \\ dn & d^2 \end{pmatrix}
$$

然而，同一作者在 [Garland 和 Heckbert 1998] 中指出，这种形式并不实用，因为它需要对更高维矩阵执行计算开销很大的矩阵运算，例如求逆。因此，本文不再进一步讨论。

```
类 QEM：IConstraint { void QEM(const IMesh*网格, const IEdge* 塌缩_
边) { auto 顶点 = 塌缩_边->获取顶点(); auto 相连_面 =
util::GetUnion( 网格->获取相连面(顶点[0]), 网格->获取相连面(顶点[1]) );
```

```
m_H = 三阶矩阵(0); m_c = 三维向量(0); m_k = 0; for (auto 面 :
相连_面) { auto 面_法向量 = util::ComputeNormal(面); auto v0 = 面->
获取顶点()[0]; double d = -点积(面_法向量, v0->获取位置()); m_H += 外
积(面_法向量, 面_法向量); m_c += d * 面_法向量; k += d *d; } }
double 评估代价(const 三维向量& v) const override { return 转置(v) * m_
H * v + 2 * 点积(m_c, v) + m_k; } }
```

5.2. 使用二次误差度量处理边界

标准二次误差度量方法在处理边界边时会遇到困难——边界边只有一个相邻面。正如 [Garland 和 Heckbert 1998], 所指出的，为解决这一问题并保留边界边，他们提出了一种改进的二次误差度量方法。

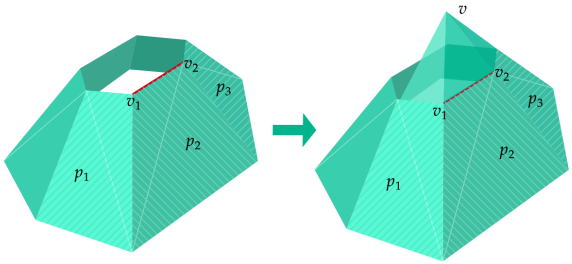

图 7。塌缩红色边时，二次误差度量会将新顶点置于相邻平面的交集处。这可能导致新顶点偏离边界

图 7 展示了标准二次误差度量方法在处理边界时的局限性。考虑红色边界边 v_1 与 v_2之间的边；在两种不同的周围几何结构下，该边被选中进行塌缩。新顶点 v 被计算为相邻平面 p_1、 p_2 和 p_3 的交集，因为该点到所有这些平面的距离均为零，因此二次误差达到可能的最小值。根据这些平面的配置，该交点可能位于原始边界的上方或下方。

在传统的二次误差度量中，没有显式约束将新顶点锚定在边界上。因此，塌缩边界边往往会使顶点偏离边界；随着更多边界边被塌缩，这种偏移会不断累积。这

这种渐进式漂移会导致网格质量明显下降，如图8所示。

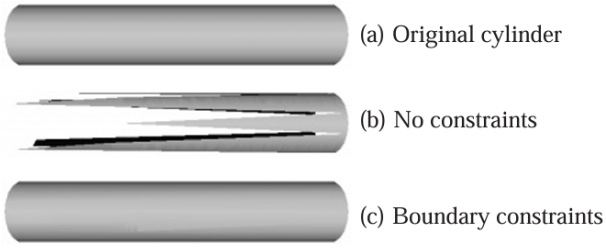

图8。有无边界约束时二次误差度量的比较

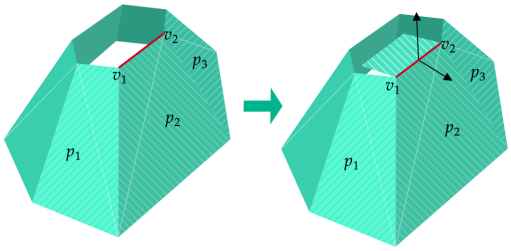

图9。为处理边界而添加的虚拟平面

为抵消这种影响，[Garland 和 Heckbert 1998] 引入了一个虚拟平面p'，如图9所示，并将其与折叠边相邻的实际平面一并纳入。p' 定义为垂直于包含边 (v_1, v_2)的网格平面的平面。一个新项 $d^2(v, p')$用于表示顶点与 p'之间的平方距离，并被纳入误差度量。随着 v 远离边界，该项会增大，从而产生将其拉回边界的修正作用。为增强这一约束，在将 p' 的二次型加到边端点的二次型上之前，先将其乘以一个较大的常数。

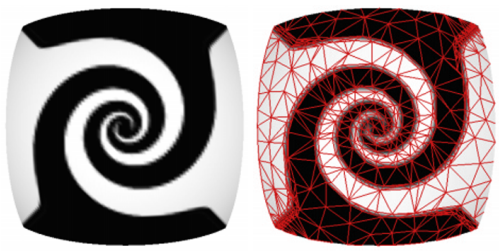

图10。采用增强型二次误差度量简化前后的带顶点颜色网格

该方法还进一步扩展，将分隔具有不同属性值（例如材质索引）的面的边视为边界。这样可以确保此类属性边界

在简化过程中得到保留，使边和面集中于这些分界处，从而改善对齐效果。此扩展的示例如图 10所示。

```
类 BoundaryQEM : 二次误差度量
{
  无效 BoundaryQEM(常量 IMesh*网格, 常量 IEdge* 塌缩_边) :
      二次误差度量(网格, 塌缩_边)
  {
    静态常量双精度 边界_二次型_比例 = 10.0;                    // 缩放
        虚拟边界平面二次型的因子

    自动 顶点 = 塌缩_边->获取顶点();
    自动 连通_面 = 求并集(
      网格->GetConnectedFaces(顶点[0])       ,
      网格->GetConnectedFaces(顶点[1])
    );

    遍历 (自动 面： 连接的_面)
    {
      自动 面_法向量 = 工具::计算法向量(面);

      遍历(常量自动 面_边： 面.获取边())
      {
          如果(!工具::是否为边界边(面_边)) 继续;

          自动 边_顶点 = 面_边->获取顶点();
          三维向量 边_位置[2] = { 边_顶点[0]->获取位置(),
              边_顶点[1]->获取位置() };
          与面_垂直的_法向量 = 归一化(叉积(面_法向量,边_位置
              [1] - 边_位置[0]));

          双精度 d = -点积(与面垂直的法向量_到_面, 边_位置[0]);
          m_H +=边界_二次型_缩放 * 外积(与面垂直的法向量_到_面,
              与面垂直的法向量_到_面);
          m_c +=边界_二次型_缩放 * d * 垂直_于_面;
          k   +=边界_二次型_缩放 * d * d;
      }
    }
  }
}
```

5.3. 体积保持约束

这项约束由 [Lindstrom 和 Turk 1998] 提出，有助于保留网格的体积。如果没有谨慎选择用于替换折叠边的新顶点，就可能使模型变形。例如，将边中点用作新顶点，可能会增大凹区域的体积，或减小凸区域的体积。此约束的目标是在每次塌缩时保留局部体积，从而尽可能减小整体体积变化

遍及整个模型。

无论是边界保持还是体积保持，都无法保证几何完整性；边界可能发生变形，曲面也可能丢失细节。然而，这些约束可以作为有用的启发式方法。保持面积和体积等简单且可量化的属性，有助于减少极端失真，即使锐边或曲线等局部特征可能会丢失。虽然这些约束无法捕捉精细的几何细节，但它们提供了一种高效的方法来维持整体结构，在无需精确保持边界或体积的复杂性的情况下，平衡准确性与性能。

当一条边 e 塌缩时，它会扫出一个四面体体积，如 图 11所示，因为每个三角形 t 都会被替换为一个新三角形 t'。

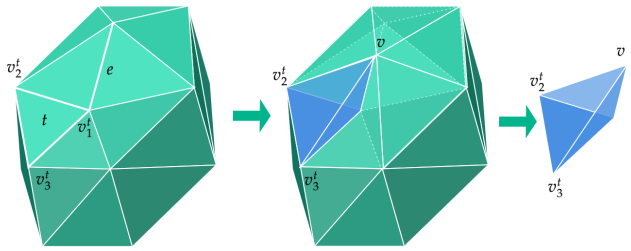

图 11。体积保持——扫出一个四面体

设 $t = [v_1, v_2, v_3]$、 $t' = [v, v_2, v_3]$以及 t 沿直线移动至v时扫出的体积为 $V(v, v_1, v_2, v_3)$。 V 在 v 位于 t 的平面上方时为正，否则为负。

因此，为了保持边折叠位置的局部体积，会考虑与边 e 相连的所有三角形 $T = \{t_1, t_2, ..., t_n\}$ 扫过的四面体的体积总和。体积变化如下所示（上标 t 表示属于三角形 t 的顶点）：

$$\begin{aligned}\mathcal{E} &= \sum_{t\in T} V\left(v, v_1^t, v_2^t, v_3^t\right) \\ &= \sum_{t\in T} \frac{1}{6}\begin{vmatrix} v_x & v_{1x}^t & v_{2x}^t & v_{3x}^t \\ v_y & v_{1y}^t & v_{2y}^t & v_{3y}^t \\ v_z & v_{1z}^t & v_{2z}^t & v_{3z}^t \\ 1 & 1 & 1 & 1 \end{vmatrix}\end{aligned}$$

求解 $\mathcal{E} = 0$ 并沿第四行展开行列式，得到：

$$0 = \frac{1}{6}\sum_{t\in T}\left\{-1\cdot\begin{vmatrix} v_{1x}^t & v_{2x}^t & v_{3x}^t \\ v_{1y}^t & v_{2y}^t & v_{3y}^t \\ v_{1z}^t & v_{2z}^t & v_{3z}^t \end{vmatrix}+1\cdot\begin{vmatrix} v_x & v_{2x}^t & v_{3x}^t \\ v_y & v_{2y}^t & v_{3y}^t \\ v_z & v_{2z}^t & v_{3z}^t \end{vmatrix}-1\cdot\begin{vmatrix} v_x & v_{1x}^t & v_{3x}^t \\ v_y & v_{1y}^t & v_{3y}^t \\ v_z & v_{1z}^t & v_{3z}^t \end{vmatrix}+1\cdot\begin{vmatrix} v_x & v_{1x}^t & v_{2x}^t \\ v_y & v_{1y}^t & v_{2y}^t \\ v_z & v_{1z}^t & v_{2z}^t \end{vmatrix}\right\}$$

将包含 v 的行列式表示为标量三重积，得到：

$$\begin{aligned} 0 &= \sum_{t\in T}\{v\cdot(v_2^t\times v_3^t)-v\cdot(v_1^t\times v_3^t)+v\cdot(v_1^t\times v_2^t)-\left|v_1^t \quad v_2^t \quad v_3^t\right|\} \\ &= v\cdot\sum_{t\in T}\left((v_2^t\times v_3^t)+(v_3^t\times v_1^t)+(v_1^t\times v_2^t)\right)-\sum_{t\in T}\left|v_1^t \quad v_2^t \quad v_3^t\right| \end{aligned} \tag{1}$$

简化项 $(v_2^t\times v_3^t+v_3^t\times v_1^t+v_1^t\times v_2^t)$，得到：

$$\begin{aligned} (v_2^t\times v_3^t+v_3^t\times v_1^t+v_1^t\times v_2^t) &= (v_2^t\times v_3^t)+(v_3^t\times v_1^t)+(v_1^t\times v_2^t)\ +(\mathbf{v_1^t}\times\mathbf{v_1^t}) \\ &= (v_2^t\times v_3^t)-(v_1^t\times v_3^t)-(v_2^t\times v_1^t)+(v_1^t\times v_1^t) \\ &= -v_2^t\times(v_1^t-v_3^t)+v_1^t\times(v_1^t-v_3^t) \\ &= (v_1^t-v_2^t)\times(v_1^t-v_3^t) \\ &= (v_2^t-v_1^t)\times(v_3^t-v_1^t) \\ &= n^t \end{aligned}$$

其中， n^t 是包含 $\triangle(v_1^t,v_2^t,v_3^t)$ 的平面的法向量，其大小等于三角形面积。

将上述简化项代入方程 1，得到：

$$\begin{aligned} 0 &= v\cdot\sum_{t\in T}n^t-\sum_{t\in T}\left|v_1^t \quad v_2^t \quad v_3^t\right| \\ \Longrightarrow v\cdot\sum_{t\in T}n^t &= \sum_{t\in T}\left|v_1^t \quad v_2^t \quad v_3^t\right| \end{aligned}$$

上述方程的形式为 $v\cdot N=D$，它定义了一个平面。这意味着向量 v 被限制在一个平面上。因此，该平面上的任意点都将满足上述方程，这表明仅靠体积保持不足以完全确定 v：我们还需要另外 2 个约束方程才能做到这一点。

```
class VolumePres : IConstraint { void VolumePres(const IMesh* 网格, const IEdge*塌缩_边) { m_H =三阶矩阵(0); m_c =三维向量(0); m_k = 0; auto 顶点 = 塌缩_边->获取顶点(); auto 相连_面 = 求并集( 网格->GetConnectedFaces(顶点[0]), 网格->GetConnectedFaces(顶点[1]) ); for (auto 面 : 相连_面) { 三维向量 面_法向量 = util::ComputeNormal(面); auto fv = 面->获取顶点(); 三维向量 位置[3] = { fv[0]->获取位置(), fv[1]->获取位置 (), fv[2]->获取位置() };
```

```
        三阶矩阵 D(0);
        D[0] = 位置[0];          D[1] = 位置[1];          D[2] =
            位置[2];

        浮点型 行列式  =行列式(D);

        m_H[0] += 面_法向量;
        米_厘[0] += det;
      }
    }

    双精度 评估代价(常量 三维向量& v) 常量 重写
    {
      返回 0;
    }
}
```

5.4. 体积优化约束

体积优化通过最小化边折叠过程中的总无符号体积变化，找到最佳的新顶点位置。它完全确定该位置，即无需额外约束。

相比之下，体积保持只确保增加的总体积与移除的总体积相互抵消，使净变化为零。正如我们所知，它会迫使顶点位于某个平面上，却不会明确指出应将顶点放在该平面的哪个位置，因此仍留有一定自由度。此外，如果在不同区域增加和减少大量体积，还可能导致局部失真。

根据体积保持约束的公式，我们知道边塌缩引起的体积变化为：

$$\varepsilon = \sum_{t \in T} V\left(v, v_1^t, v_2^t, v_3^t\right)$$

其中：

- $t = \triangle\left(v_1^t, v_2^t, v_3^t\right)$
- $t' = \triangle(v, v_2, v_3)$
- $V(v, v_1, v_2, v_3)$ 是 v_1 沿线性路径移动至 v时， t 扫过的体积

如果顶点 v 位于三角形 t所在平面的上方，则四面体的有向体积 V 为正；如果位于下方，则为负。但对于优化而言，我们关注的是体积变化的大小，而不是方向。因此，我们使用无符号体积变化。为了得到无符号体积，我们可以使用 $|V|$ 或 V^2。我们使用 V^2，因为它处处可微。这一点很重要，因为优化算法依赖梯度，而 $|V|$ 在零点处存在折角，此处的梯度未定义。

因此，我们改用体积平方和来表示 $\mathcal{E}$。于是得到：

$$\begin{aligned}
\mathcal{E} &= \sum_{t\in T} V\left(v,\, v_1^t,\, v_2^t,\, v_3^t\right)^2 \\
&= \sum_{t\in T} \left(v\cdot n_t - \left|v_1^t \quad v_2^t \quad v_3^t\right|\right)^2 \\
&= \sum_{t\in T} \left(\left(v\cdot n_t\right)^2 + \left|v_1^t \quad v_2^t \quad v_3^t\right|^2 - 2v\cdot n_t \left|v_1^t \quad v_2^t \quad v_3^t\right|\right) \\
&= \left(\sum_{t\in T} v^T\left(n_t n_t^T\right)v\right) - 2\left(\sum_{t\in T}\left(\left|v_1^t \quad v_2^t \quad v_3^t\right| n_t\right)\cdot v\right) + \sum_{t\in T}\left|v_1^t \quad v_2^t \quad v_3^t\right|^2 \\
&= v^T\left(\sum_{t\in T}\left(n_t {n_t}^T\right)\right)v + 2\left(-\sum_{t\in T}\left(\left|v_1^t \quad v_2^t \quad v_3^t\right| n_t\right)\right)\cdot v + \sum_{t\in T}\left|v_1^t \quad v_2^t \quad v_3^t\right|^2 \\
&= v^T H v + 2c^T v + k
\end{aligned}$$

于是我们得到：

$$v = -H^{-1}c$$

该约束可以唯一确定 v，退化情况除外，此时 $\det(H) = 0$。与二次误差度量的情况一样，这种情况会出现在几何体的局部平坦区域，因为我们知道 H 定义如下：

$$H = \sum_t \left(n_t {n_t}^T\right)$$

其形式与二次误差度量中使用的形式相同。因此，在所有 n_t 都彼此平行、总和简化为缩放后的秩一项时，H 在平坦区域中会变得不可逆。在这种情况下，需要采用其他约束或回退策略来进行顶点放置。

```
class VolumeOpt : IConstraint { void VolumeOpt(const IMesh* mesh, const
IEdge*塌缩_边) { auto 顶点 = 塌缩_边->获取顶点(); auto 相连_面 = 求并集( 网
格->获取相连面(顶点[0]), 网格->获取相连面(顶点[1]) ); m_H =三阶矩阵(0); m_c =
三维向量(0); m_k = 0; for (const auto face : 相连_面) { 三维向量 面_法向量 =
util::ComputeNormal(face); auto fv = 面->获取顶点();
```

```
        三维向量 位置[3] = { fv[0]->获取位置(),                    fv[1]->获取位置
            (), fv[2]->获取位置()        };

        三阶矩阵 D(0);
        D[0] = 位置[0];          D[1] = 位置[1];          D[2] =
            位置[2];

        float det   =行列式(D);

        m_H   += 外积(面_法向量, 面_法向量);
        m_c   += det* 面_法向量;
        米_千  += 确定* 确定;
      }

      m_c = -1.0 * m_c;
    }

    双精度 评估代价(常量 三维向量& v) 常量 重写
    {
      返回 转置(v) * m_H *v   + 2.0 * 点积(m_c, v)   + m_k;
    }
}
```

5.5. 边界保持约束

该约束在 [Lindstromand Turk 1998]中进行了讨论。它有助于通过保持边界的面积来计算最优顶点放置位置和边折叠误差。

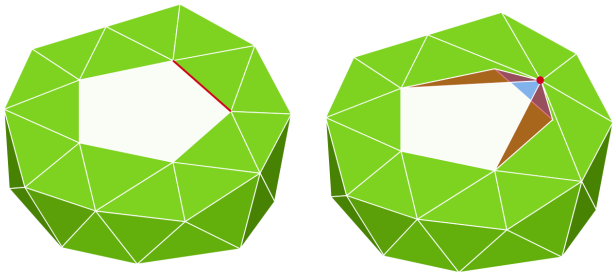

图12。边界保持示例（面积未按比例绘制）

在图12中，左图显示了平面孔洞上的一条边界边（红色），而右图显示了边折叠后该边界边被新顶点（红点）替代的情况。因此，尽管着色区域的总面积得以保留（红色区域的面积损失由蓝色区域的面积增益抵消），边界形状和结构却发生了明显改变。因此，该约束保留的是面积，而不是边界本身。

根据图13，设边 (v_1, v_2) 塌缩至顶点 v，并设 ε 为净面积变化。塌缩一条边界边会将新顶点 v 连接到另外两条边界

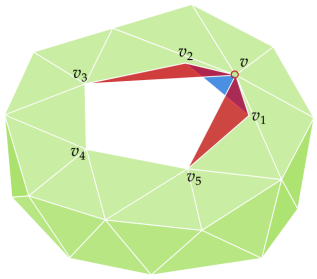

图13。边界保持——平面边界情形

边，从而形成三个三角形。它们的有向面积之和得出 $\mathcal{E}$。

$$\begin{aligned}\mathcal{E} &= Area(v, v_1, v_2) + Area(v, v_2, v_3) + Area(v, v_5, v_1) \\ &= \frac{1}{2}\begin{pmatrix}(v \times v_1) + (v_1 \times v_2) + (v_2 \times v) + \\ (v \times v_2) + (v_2 \times v_3) + (v_3 \times v) + \\ (v \times v_5) + (v_5 \times v_1) + (v_1 \times v)\end{pmatrix}\end{aligned}$$

虽然平面边界有助于直观理解边折叠过程中的面积变化，但实际边界通常并非平面。不过，正如附录（第9节）中详细说明的那样，同样的表述方式也适用于非平面边界。

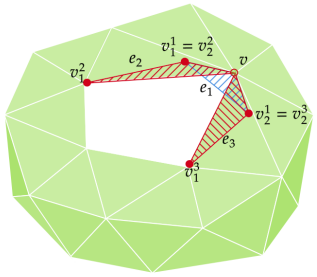

图14。边界保持——约束推导

令 $\mathcal{E}$ 表示边折叠导致的面积变化的平方，$E = \{e_1, \dots, e_n\}$ 表示与顶点 v 相连的边界边集合，其中每条边为 $e : (v_1^e, v_2^e) \in E$。

折叠一条边界边会连接另外两个顶点，形成三个三角形（$n = 3$），如图14所示。采用平方范数是为了简化计算。随后，将顶点 v 引起的总面积变化的平方定义为误差：

$$\begin{aligned}\mathcal{E} &= \left\|\sum_{e\in E} Area\left(v, v_1^e, v_2^e\right)\right\|^2 \\ &= \left\|\sum_{e\in E} \frac{1}{2}\left(\left(v\times v_1^e\right)+\left(v_1^e\times v_2^e\right)+\left(v_2^e\times v\right)\right)\right\|^2 \\ &= \left\|\sum_{e\in E} \frac{1}{2}\left(v\times\left(v_1^e-v_2^e\right)+\left(v_1^e\times v_2^e\right)\right)\right\|^2 \\ &= \frac{1}{4}\left\|v\times\sum_{e\in E}\left(v_1^e-v_2^e\right)+\sum_{e\in E}\left(v_1^e\times v_2^e\right)\right\|^2\end{aligned}$$

为简化起见，将 $\sum_{e\in E}\left(v_1^e-v_2^e\right)$ 记作 E_1，将 $\sum_{e\in E}\left(v_1^e\times v_2^e\right)$ 记作 E_2。于是，

$$\mathcal{E} = \frac{1}{4}\left\|v\times E_1+E_2\right\|^2 = \frac{1}{4}\left\|E_2-E_1\times v\right\|^2$$

为了简化叉积项，$E_1\times v$ 可以写成 $\mathcal{S}v$，其中 $\mathcal{S}$ 是与 E_1进行叉积运算的反对称矩阵：

$$\mathcal{E} = \frac{1}{4}\left\|E_2-\mathcal{S}v\right\|^2$$

化简平方范数可得：

$$\begin{aligned}\mathcal{E} &= \frac{1}{4}(E_2-\mathcal{S}v)^T(E_2-\mathcal{S}v) \\ &= \frac{1}{4}\left(E_2^T-v^T\mathcal{S}^T\right)(E_2-\mathcal{S}v) \\ &= \frac{1}{4}\left(v^T\mathcal{S}^T\mathcal{S}v+2\left(-{E_2}^T\mathcal{S}\right)^T v+E_2^TE_2\right) \\ &= \frac{1}{4}\left(v^THv+2c^Tv+k\right)\end{aligned}$$

其形式与二次误差度量（QEM）中的相同，这表明解应当相同：$v=-H^{-1}c$。

然而，对于此约束，由于 $\mathcal{S}$ 是反对称矩阵，因此 H 不可逆。因此，无法完全确定 v。下面演示将梯度设为零后实际能够得到的约束。

$$\begin{aligned}\nabla\mathcal{E} &= \frac{1}{4}(2Hv+2c) \\ \Longrightarrow 0 &= \frac{1}{4}\left(2\mathcal{S}^T\mathcal{S}v+2\mathcal{S}E_2\right) \\ \Longrightarrow 0 &= \mathcal{S}E_2-\mathcal{S}^2v \\ &= \mathcal{S}(E_2-\mathcal{S}v) \\ &= E_1\times(E_2-E_1\times v) \\ &= E_1\times E_2-(E_1\times(E_1\times v)) \\ &= E_1\times E_2-E_1(E_1\cdot v)+v(E_1\cdot E_1) \\ &= v\left(\|E_1\|^2\right)-E_1(E_1\cdot v)+K\end{aligned}$$

... $\mathcal{S}^{\mathrm{T}}=-\mathcal{S}$对于反对称矩阵... $\mathcal{S}$，由于 0 不可逆，因此不能与 0 相消.....使用向量三重积.....将 $E_1\times E_2$ 记为 K

这就得到：

$$0 = v\left(\|E_1\|^2\right) - E_1(E_1 \cdot v) + K \tag{2}$$

这是 v 中的三维向量方程，可以将其分裂为 3 个标量方程，以求解其各个分量。我们可以为此选择任意基，但为方便起见，我们选择：$[K, E_1, E_1 \times K]$。

将方程 2投影到 K 可得：

$$\begin{aligned}
0 &= K \cdot [v(E_1 \cdot E_1) - E_1(E_1 \cdot v) + K] \\
&= K \cdot v\left(\|E_1\|^2\right) - K \cdot E_1(E_1 \cdot v) + K \cdot K \\
&= \left[K \cdot v\left(\|E_1\|^2\right)\right] - \cancel{[\mathbf{K} \cdot \mathbf{E_1}(\mathbf{E_1} \cdot \mathbf{v})]} + K \cdot K \quad \dots K = (E_1 \times E_2) \perp E_1 \; so \; K.E_1 = 0 \\
&= (E_1 \times E_2) \cdot \left(v\left(\|E_1\|^2\right)\right) + \|E_1 \times E_2\|^2 \\
&= \|E_1\|^2 (E_1 \times E_2) \cdot v + \|E_1 \times E_2\|^2
\end{aligned}$$

将方程 2 投影到 E_1 可得：

$$\begin{aligned}
0 &= E_1 \cdot [v(E_1 \cdot E_1) - E_1(E_1 \cdot v) + K] \\
&= \cancel{\mathbf{E_1} \cdot \mathbf{v}(\mathbf{E_1} \cdot \mathbf{E_1})} - \cancel{\mathbf{E_1} \cdot \mathbf{E_1}(\mathbf{E_1} \cdot \mathbf{v})} + E_1 \cdot (E_1 \times E_2) \\
&= E_1 \cdot (E_1 \times E_2) \\
&= 0 \qquad \dots E_1 \perp (E_1 \times E_2)
\end{aligned}$$

这简化为 $0 = 0$，这是一个退化结果。这表明，v 平行于 E_1的分量无法通过该最小化问题确定，因为它不会影响误差函数的值。

将 方程 2 投影到 $E_1 \times K$ 可得：

$$\begin{aligned}
0 &= (E_1 \times K) \cdot [v(E_1 \cdot E_1) - E_1(E_1 \cdot v) + K] \\
&= (E_1 \times K) \cdot v\left(\|E_1\|^2\right) - (E_1 \times K) \cdot E_1(E_1 \cdot v) + \cancel{(E_1 \times K) \cdot K} \qquad \dots K \perp (E_1 \times K) \\
&= (E_1 \times K) \cdot v\left(\|E_1\|^2\right) - \cancel{(E_1 \times K) \cdot E_1(E_1 \cdot v)} \qquad \dots E_1 \perp (E_1 \times K) \\
&= (E_1 \times (E_1 \times E_2)) \cdot v\left(\|E_1\|^2\right) \\
&= (E_1 \times (E_1 \times E_2)) \cdot v
\end{aligned}$$

因此，此优化的解空间位于由以下条件定义的两个平面的交集中：

$$\begin{aligned}
\|E_1\|^2 (E_1 \times E_2) \cdot v + \|E_1 \times E_2\|^2 &= 0 \\
(E_1 \times (E_1 \times E_2)) \cdot v &= 0
\end{aligned}$$

即使采用这种方法，ε、v 仍未确定。因此，还需要额外的约束以及两个方程，才能求解 v。

```
类 BoundaryPres : IConstraint{void BoundaryPres(const IMesh*mesh,
const IEdge* 塌缩_边){
```

```
if(!util::IsBoundaryEdge(collapse_边)) return; m_H =mat3(0); m_c = vec3(0); m_k = 0; auto 顶点 = 塌缩_边->获取顶点(); auto connected_边 = 求并集( mesh->GetConnectedEdges(顶点[0]), mesh->GetConnectedEdges(顶点[1]) ); vec3 E1(0); vec3 E2(0); for (auto edge : connected_边) { if(!util::IsBoundaryEdge(edge)) continue; const auto verts = 边->获取顶点(); vec3 位置[2] = { verts[0]->获取位置(), verts[1]->获取位置() }; E1 += 位置[0] - 位置[1]; E2 += 叉积(位置[0], 位置[1]); } mat3 skew_对称_矩阵 = util::构造反对称矩阵(E1); H +=转置(skew_对称_矩阵) * skew_对称_矩阵; c += skew_对称_矩阵 *E2; k += 点积(E2, E2); } double 评估代价(const vec3& v) const override { return 0.25 * (转置(v) * m_H *v) + 0.5 * 点积(m_c, v) +0.25* m_k; } }
```

5.6. 边界优化约束

Lindstrom 和 Turk 提出的边界优化约束 [Lindstrom 和 Turk 1998]与边界保持类似，会最小化边界三角形面积，但侧重于无符号面积。误差表示为有符号面积平方之和，即：

$$\mathcal{E} = \sum_{e \in E} A\left(v, v_1^e, v_2^e\right)^2 \tag{3}$$

其中，

- $v \in \mathbb{R}^3$

- v_1^e, v_2^e = 每条边界边的端点 $e \in E$
- $A(v, v_1^e, v_2^e)$ = 由 $\triangle(v, v_1^e, v_2^e)$形成的面积

展开公式 3 可得:

$$\mathcal{E} = \frac{1}{4}\sum_{e\in E}\left\|v\times(v_1^e - v_2^e) + (v_1^e \times v_2^e)\right\|^2$$

代入 $e_1 = (v_1^e - v_2^e)$ 和 $e_2 = (v_1^e \times v_2^e)$, 可得:

$$\begin{aligned}\mathcal{E} &= \frac{1}{4}\sum_{e\in E}\left\|v\times e_1 + e_2\right\|^2 \\ &= \frac{1}{4}\sum_{e\in E}\left(\left\|v\times e_1\right\|^2 + \left\|e_2\right\|^2 + 2(v\times e_1)\cdot e_2\right) \\ &= \frac{1}{4}\left(\sum_{e\in E}\left(\left\|v\times e_1\right\|^2 + 2(v\times e_1)^T e_2\right) + \sum_{e\in E}\left\|e_2\right\|^2\right)\end{aligned}$$

为简化叉积项, $v\times e_1$ 可写为 $\mathcal{S}v$, 其中 $\mathcal{S}$ 是与 e_1做叉积的反对称矩阵。此外, 可以利用标量三重积恒等式重新排列 $2(v\times e_1)^T e_2 = 2(e_1\times e_2)^T v$。

$$\begin{aligned}\mathcal{E} &= \frac{1}{4}\left(\sum_{e\in E}\left(\left\|\mathcal{S}v\right\|^2 + 2(e_1\times e_2)^T v\right) + \sum_{e\in E}\left\|e_2\right\|^2\right) \\ &= \frac{1}{4}\left(\sum_{e\in E}\left(v^T\left(\mathcal{S}^T\mathcal{S}\right)v\right) + 2\left(\sum_{e\in E}(e_1\times e_2)^T\right)v + \sum_{e\in E}\left\|e_2\right\|^2\right) \\ &= \frac{1}{4}\left(v^T\left(\sum_{e\in E}\mathcal{S}^T\mathcal{S}\right)v + 2\left(\sum_{e\in E}(e_1\times e_2)\right)^T v + \sum_{e\in E}\left\|e_2\right\|^2\right) \\ &= \frac{1}{4}\left(v^T H v + 2c^T v + k\right)\end{aligned}$$

其形式与前面的约束相同, 因此我们得到 v 为 $v = -H^{-1}c$。

与前面的约束一样, 如果 $\det(H) = 0$, 该约束就会退化, 我们需要使用其他约束或采用备用策略来放置顶点。

```
class BoundaryOpt : IConstraint { void BoundaryOpt(const Mesh* mesh, const
IEdge*塌缩_边) { if(!util::IsBoundaryEdge(塌缩_边)) return; auto 顶点 = 塌缩_
边->获取连通顶点(); auto 连通_边 = 求并集( mesh->GetConnectedEdges(顶点[0]),
mesh->GetConnectedEdges( 顶点[1]) ); m_H = 三阶矩阵 (0); m_c = 三维向量
(0); m_k = 0; for (auto 边 : 连通_边) { if(!util::IsBoundaryEdge(边)) continue;
```

```
      const auto 顶点 = 边->获取顶点();
      三维向量 位置[2] = { 顶点[0]->获取位置(),                    顶点[1]->
          获取位置()};

      三维向量 e1 = 位置[0] - 位置[1];
      三维向量 e2 = 叉积(位置[0],          位置[1]);

      三阶矩阵 skew_sym_mat = util::构造反对称矩阵(E1);

      m_H +=转置(skew_sym_mat) * skew_sym_mat;

      m_c  += 叉积(e1,   e1);
      m_k  += 点积(e2,   e2);
    }
  }

  double 评估代价(const 三维向量& v) const override
  {
    return 0.25 * (转置(v) * m_H *v)  + 0.5 * 点积(m_c,       v) +
        0.25 * m_k;
  }
}
```

5.7. 三角形形状优化约束

三角形形状优化旨在提升三角形质量。细长或拉伸的三角形可能导致着色问题，而形状更均匀、接近等边三角形的网格看起来更美观，效果也更好。

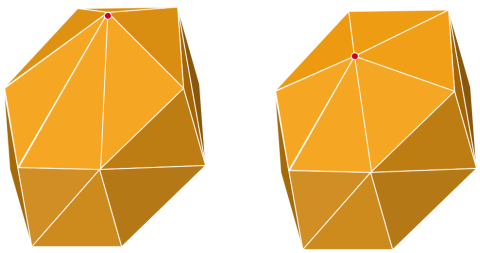

图15。产生低质量（左）与高质量（右）三角形形状的边折叠

如图15所示，左侧的顶点放置形成了更规整、形状更均匀的三角形，因此三角形结构更加整洁。右侧的放置则包含细长的三角形，使网格显得不够整齐，视觉效果也较差。

在进一步分析之前，需要先介绍一个重要的三角形形状质量指标：**面积周长比**。正则三角形（如等边三角形）的

面积周长比更高，这意味着它们更加紧凑，不那么细长。

在三角形形状保持约束中，目标是最大化面积周长比。我们假设折叠边周围的区域近乎平坦。这意味着边折叠后，附近三角形的总面积不会发生太大变化。因此，为了提高它们的面积周长比，我们可以只专注于减小周长，而周长由边长决定。

这就是我们要最小化与新顶点相连的边的长度平方之和的原因。这会将顶点拉到一个位置，使各边分布得更加均匀且更短，从而形成更均衡、不那么狭长的三角形。

我们将此误差 ε 表示为：

$$\varepsilon = \sum_{i=1}^{n} ||v - v_i||^2$$

其中， v_i 表示网格中与 v 相连的每个邻近顶点 $\{v_1, \dots, v_n\}$。展开该方程以求 ε，可得：

$$\begin{aligned}
\varepsilon &= \sum_{i=1}^{n} ||v - v_i||^2 \\
&= \sum_{i=1}^{n} \left(||v||^2 + ||v_i||^2 - 2v \cdot v_i \right) \\
&= \sum_{i=1}^{n} ||v||^2 + \sum_{i=1}^{n} ||v_i||^2 - 2v \cdot \sum_{i=1}^{n} v_i \\
&= \sum_{i=1}^{n} v^T v + \sum_{i=1}^{n} ||v_i||^2 - 2v \cdot \sum_{i=1}^{n} v_i \\
&= v^T \left(\sum_{i=1}^{n} I \right) v + 2v \cdot \left(-\sum_{i=1}^{n} v_i \right) + \sum_{i=1}^{n} ||v_i||^2 \quad \dots I \text{ is an identity matrix} \\
&= v^T H v + 2c^T v + k
\end{aligned}$$

可以像处理其他约束一样求解。此处得到的解为：

$$v = -H^{-1}c = -\left(\frac{1}{n} I\right)\left(-\sum_{i=1}^{n} v_i\right) = \frac{1}{n}\sum_{i=1}^{n} v_i$$

因此，优化三角形形状会使我们选择相邻顶点的质心。

避免瘦长三角形是因为它们会导致着色伪影。光栅化会使用重心坐标对逐顶点数据（例如法线）进行插值，其定义如下：

$$u = \frac{Area(\triangle ABP)}{Area(\triangle ABC)}, \quad v = \frac{Area(\triangle APC)}{Area(\triangle ABC)}, \quad w = \frac{Area(\triangle BCP)}{Area(\triangle ABC)}$$

对于瘦长三角形，分母 $Area(\triangle ABC)$ 会变得非常小，导致坐标在数值上不稳定。顶点位置或子区域面积中的微小浮点误差，随后可能造成较大的插值误差，从而产生着色伪影。

```
类 TriShapeOpt : IConstraint{void TriShapeOpt(const IMesh*网格,
const IEdge* 塌缩_边){自动 顶点 = 塌缩_边->获取顶点();
```

```
    自动 连通_顶点 = 求并集(
      网格->获取连通顶点(顶点[0])                    ,
      网格->获取连通顶点(顶点[1])
    );

    m_H = 三阶矩阵(0); m_c = 三维向量(0);     m_k = 0;

    遍历 (常量 auto 顶点： 连接的_顶点)
    {
      m_H += 三阶矩阵(1); //单位矩阵
      三维向量 位置 = 顶点.获取位置();
      m_c += -pos;
      m_k += 点积(位置, 位置);
    }
  }

  双精度 评估代价(常量 三维向量& v) 常量 重写
  {
    返回 转置(v) * m_H *v + 2.0 * 点积(m_c, v) + m_k;
  }
}
```

5.8. 回退策略

当用于计算 v 的矩阵不可逆时，需要采用回退策略。一种简单且常见的回退方法是将新顶点放置在正在塌缩的边 (v_1, v_2)的中点：

$$v = \frac{v_1 + v_2}{2}$$

```
三维向量 获取回退顶点(IEdge* 塌缩_边){自动 顶点 = 塌缩_边->获取顶点();返回
0.5 *(顶点[0]->获取位置() + 顶点[1]->获取位置());}
```

6. 约束选择标准

为求解新顶点 v，根据前面各节的讨论，由若干约束构成一个线性方程组。每个线性方程的形式如下：

$$a^T v = b$$

这里，每个 a 表示一个约束平面的法向量，b 是对应的偏移量。从几何角度看，我们要找的是位于 $\mathbb{R}^3$中所有这些平面交集处的点 v。

理论上，只需要三个线性独立的约束（平面），就能唯一确定三维空间中的一个点。然而，该算法纳入了三个以上的约束，以确保稳健性。这是因为：

- 某些约束可能会变得冗余（线性相关）。
- 某些约束可能会在平坦或对称区域中退化。

最终的顶点放置只包含通过以下标准逐一添加约束并检查线性无关性和稳定性后筛选出的最佳三个约束：**第一个约束(a_1)：是否有效?**

$$a_1 \neq 0$$

```
布尔 IsFirstConstraintValid(常量 三维向量 &A, 常量 三阶矩阵 &H, 常量 三维向量&c)

{
    返回 长度(A)  > 0;
}
```

第二约束(a_2)：它是否与第一个约束线性无关?

$$(a_1 \cdot a_2)^2 < (\|a_1\| \, \|a_2\| \cos(\alpha))^2$$

此项用于检查第一个和第二个约束的法线之间的角度 θ 是否足够大，也就是说，它们是否并非近乎平行。 α 是用于确定线性无关性是否可接受的阈值角。

```
布尔值 IsSecondConstraintValid(常量 三维向量 &A, 浮点数 alpha, 常量 三阶矩
阵&H, 常量 三维向量 &c){

    浮点数 lhs  =点积(H[0], A);浮点数 rhs  = 长度(H[0]) *
    长度(A)* 余弦(alpha);返回 lhs* lhs  < rhs * rhs;

}
```

第三个约束(a_3)：它是否不与前两个共面?

$$((a_1 \times a_2) \cdot a_3)^2 < \left(\|a_1 \times a_2\| \, \|a_3\| \, \sin(\alpha)\right)^2$$

此项确保第三个约束的法向量 a_3 不位于由前两个约束的法线构成的平面内，同时也允许一定的阈值角 α。这样可确保三个平面在三维空间中交于一点，从而为新顶点确定唯一解。

请注意，在计算点积时，我们在此使用 $\sin(\alpha)$，而不是 $\cos(\alpha)$。我们将 α 定义为由 a_1 和 a_2 形成的平面与新向量 a_3 之间的角度。因此，

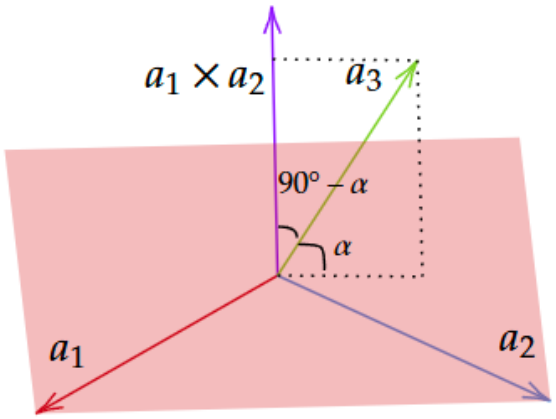

图 16。3 个约束法向量的可视化

图 16 表明，向量 $a_1 \times a_2$ 和 a_3 之间的角度将为 $90° - \alpha$。因此，我们有：

$$\begin{aligned}(a_1 \times a_2) \cdot a_3 &= \|a_1 \times a_2\| \, \|a_3\| \cos(90° - \alpha) \\ &= \|a_1 \times a_2\| \, \|a_3\| \sin(\alpha)\end{aligned}$$

```
布尔值 IsThirdConstraintValid(常量 三维向量 &A, 浮点数 alpha, 常量 三阶矩阵 &
    H, 常量 三维向量 &c)
{
    vec3 p = 叉积(H[0],   H[1]);

    浮点数 左侧  =点积(p,   A);
    浮点数 右侧  =长度(p) * 长度(A) * 正弦(alpha);
    返回 lhs* lhs  < rhs * rhs;
}
```

7. 顶点放置

综合以上内容，我们重新审视整体边折叠算法，并实现其所有组件。首先，我们按照最适合当前用例的顺序计算上述约束。接下来，应用约束选择标准，形成一个可解约束集合。最后，根据该集合计算最优顶点位置；如果系统仍然无解，则采用回退策略。随后返回计算得到的最优位置及其对应代价，以便在优先队列中进一步处理。

```
void获取塌缩顶点(constIMesh* 网格,constIEdge&塌缩_边, 三维向量 &塌缩_顶点,
float & 塌缩 _ 误差 ){ 向量 <IConstraint>  约束 ;
```

```
IConstraint 二次误差度量 = 二次误差度量(网格, 塌缩_边); 约束.add(二次误差度量);
IConstraint 体积_保持 = VolumePres(网格, 塌缩_边); 约束.add(体积_保持);
IConstraint 体积_优化 =VolumeOpt(网格, 塌缩_边); 约束.add(体积_优化);
IConstraint 边界_二次误差度量 = BoundaryQEM(网格, 塌缩_边); 约束.add(边界_二次
误差度量); IConstraint 边界_保持 = BoundaryPres(网格, 塌缩_边); 约束.add(边界_保
持); IConstraint 边界_优化 = BoundaryOpt(网格, 塌缩_边); 约束.add(边界_优化);
IConstraint 三角形_优化 = TriangleOpt(网格, 塌缩_边); 约束.add(三角形_优化); int
n_方程 = 0; floatalpha =弧度(5.0); //5度,以弧度表示 mat3 H; 三维向量 c; for (auto 约
束 : 约束) { if (n_方程 == 3) break; mat3constr_H =约束.获取 H(); 三维向量constr_
c =约束. 获取 C(); for(int i = 0; i < 3; i++) { 三维向量 A = constr_H [i]; float b =
constr_C [i]; if (n_方程 == 0 && IsFirstConstraintValid(A, H, c)) { H[0] =A; c
[0] =b; n_方程++; } else if (n_方程 == 1 && IsSecondConstraintValid(A, alpha,
H, c )) { H[1] =A; c[1] =b; n_方程++; } else if (n_方程 == 2 &&
IsThirdConstraintValid(A, alpha, H, c) ) { H[2] =A; c[2] =b; n_方程++; break; } }
```

```
} 塌缩_顶点 = 行列式(H) == 0.0 ? 获取回退顶点( 塌缩_边) : inverse(H) *
c; 塌缩_误差 = 0; for (auto 约束 : 约束) { // 注意：这里可以对“评估代价”函数进行
加权。 // 权重可以由用户定义，以优先处理 // 不同类型的网格变形最小化 // （例如，
优先保持体积 // 而非三角形质量）塌缩_误差 += 约束.EvaluateCost(塌缩_误差); }
```

8. 处理网格属性

到目前为止，本文的重点一直仅放在简化过程中三维网格的几何体和拓扑上。然而，现实世界中的网格通常还包含逐顶点颜色和法线等属性，或渲染时使用的逐面材质索引。通过边折叠简化此类网格时，务必保持新顶点处的属性一致性，以避免颜色接缝等视觉伪影。以下各节将探讨应对这一挑战的关键方法。

8.1. 高维二次曲面用于二次误差度量

[Garland 和 Heckbert 1998] 提出了对二次误差度量方法的改进，在边折叠过程中计算新顶点时，除了位置之外，还纳入顶点属性。该方法使用高维二次曲面，其中每个顶点都表示为高维空间中的一个向量 $\mathbb{R}^n$，$n > 3$将其位置与所有相关属性结合起来。例如，如果一个顶点 (x, y, z) 只有颜色属性(R, G, B)，则可将其表示为向量 $(x, y, z, R, G, B) \in \mathbb{R}^6$。

即使顶点坐标给定在三维以上的空间中，仍可使用二次误差度量（新点到相邻平面的距离之和）。现在，计算到每个平面的距离时使用的是 n 维坐标，但所对应的仍是二维平面，因为无论空间维数有多少，三个非共线点都能定义一个平面。

要计算点 v 到平面的距离，我们首先确定一个从 v 指向该平面的向量 u，使其垂直于平面。为此，我们定义一个标准正交基，其中两个基向量张成该平面，并将它们记为 e_1 和 e_2。这两个向量属于一个完整的标准正交基 $\{e_1, e_2, \ldots, e_n\}$，该基适用于整个 $\mathbb{R}^n$。无需知道 $\{e_3, \ldots, e_n\}$ 的确切值；我们只需知道这样的基存在即可。

如有需要，可以使用格拉姆－施密特正交化过程确定它们。该过程从 n 个线性无关向量开始，并通过迭代去除与先前计算出的基向量平行的分量。

由三个点 p、q、$r \in \mathbb{R}^n$ 确定一个平面。

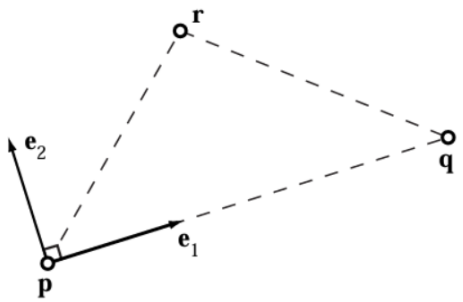

图17。 $\mathbb{R}^n$ 中的局部参考坐标系

如图17所示，可在由 (p,q,r)张成的平面上定义两条正交轴 e_1 和 e_2：

- e_1 是沿着方向 $q-p$ 的单位向量，即 $e_1 = \dfrac{q-p}{\|q-p\|}$
- 使用格拉姆－施密特正交化过程，可通过取向量 $r-p$，减去其在 e_1 上的投影（以确保与 e_1 正交），然后进行归一化，得到 e_2。换言之，

$$e_2 = \frac{(r-p) - (e_1 \cdot (r-p))e_1}{\|(r-p) - (e_1 \cdot (r-p))e_1\|}$$

接下来，选择平面上的一点 p，并将从 p 到 v 的向量，即 $w = v-p$，表示为其沿标准正交基 $\{e_1, e_2, \ldots, e_n\}$ 的分量之和：

$$w = \sum_{i=1}^{n} (w \cdot e_i) e_i$$

接下来，从 w 中去掉沿平面的分量，即 (e_1, e_2)，得到向量 u：

$$u = w - (w \cdot e_1)e_1 - (w \cdot e_2)e_2 = v - ((w \cdot e_1)e_1 + (w \cdot e_2)e_2 + p)$$

这里，u 是从 v 到平面的垂线，其长度平方恰好为 ε：

$$
\begin{aligned}
\mathcal{E} &= u \cdot u \\
&= (w - (w \cdot e_1)e_1 - (w \cdot e_2)e_2) \cdot (w - (w \cdot e_1)e_1 - (w \cdot e_2)e_2) \\
&= \|w\|^2 + (w \cdot e_1)^2(e_1 \cdot e_1) + (w \cdot e_2)^2(e_2 \cdot e_2) && \dots e_1 \cdot e_1 = e_2 \cdot e_2 = 1 \\
&\quad - 2(w \cdot e_1)^2 - 2(w \cdot e_2)^2 + \cancel{2(w \cdot e_1)(w \cdot e_2)(e_1 \cdot e_2)} && \dots e_1 \cdot e_2 = 0 \\
&= \|w\|^2 - (w \cdot e_1)^2 - (w \cdot e_2)^2
\end{aligned}
$$

使用 $w = v - p$ 进一步展开后，我们得到：

$$
\begin{aligned}
\mathcal{E} &= (v - p) \cdot (v - p) - ((v - p) \cdot e_1)^2 - ((v - p) \cdot e_2)^2 \\
&= (v \cdot v + p \cdot p - 2p \cdot v) - (e_1 \cdot v - e_1 \cdot p)^2 - (e_2 \cdot v - e_2 \cdot p)^2 \\
&= \left(v \cdot v - (e_1 \cdot v)^2 - (e_2 \cdot v)^2\right) + 2(-p \cdot v + (e_1 \cdot p)(e_1 \cdot v) + (e_2 \cdot p)(e_2 \cdot v)) + \left(p \cdot p - (e_1 \cdot p)^2 - (e_2 \cdot p)^2\right) \\
&= \left(v^T v - v^T e_1 e_1^T v - v^T e_2 e_2^T v\right) + 2((p \cdot e_1)e_1 + (p \cdot e_2)e_2 - p) \cdot v + \left(p \cdot p - (p \cdot e_1)^2 - (p \cdot e_2)^2\right) \\
&= v^T \left(I - e_1 {e_1}^T - e_2 {e_2}^T\right) v + 2((p \cdot e_1)e_1 + (p \cdot e_2)e_2 - p)^T v + \left(p \cdot p - (p \cdot e_1)^2 - (p \cdot e_2)^2\right) \\
&= v^T H v + 2c^T v + k
\end{aligned}
$$

请注意，此表达式与原始二次误差度量方法中使用的误差形式相符。这意味着，除了实体 $\{H, c, k\}$ 的数值和维度有所不同之外，计算代价和确定最优 v 的过程仍保持不变：最优 v 仍由 $-H^{-1}c$ 给出。

此外，计算得出后，v 不仅表示最优顶点位置，还编码了所有相关标量属性的最优值。

8.2 基于能量的代价计算

[Hoppe 1996]提出了渐进网格——由一系列网格组成，用于表示输入网格不同细节层次；这些网格均通过连续执行边折叠操作生成。他们提出了一种计算每次边折叠代价的替代方法，将其定义为能量函数的差值。该代价反映了塌缩前后网格能量函数的变化程度；能量差较小的边被视为更适合塌缩的候选边。他们的方法还处理每个细节层次上的离散面属性和标量顶点属性。

8.2.1. 前置条件

在介绍成本函数之前，我们先定义计算边折叠代价所需的关键几何实体和准备步骤。图18展示了一个示例。

原始网格定义：

将进行任何边折叠操作之前的原始网格记作：

$$\hat{M} = (\hat{V}, \hat{F}, \underline{\hat{V}}, \underline{\hat{F}})$$

- **顶点：** $\hat{V} = \{v_1, \dots, v_n\}$，其中每个 $v_i \in \mathbb{R}^3$

- **面：** $\hat{F} = \{f_1, \ldots, f_m\}$，其中每个 $f_i \in \{1, \ldots, n\}^3$
- **顶点属性：** $\hat{V} = \{v_1, \ldots, v_n\}$，每个 $v_i \in \mathbb{R}^d$（例如，$d = 3$ 用于 RGB颜色）
- **面属性：** $\hat{F} = \{f_1, \ldots, f_m\}$，每个 $f_i \in \mathbb{Z}^{d'}$（例如，$d' = 1$ 用于材质索引）

简化设置：

简化前，会执行以下步骤：

1. **表面采样：**

   在 $\hat{M}$ 的曲面上采样一个由 k 个点组成的集合 X：
   $X = \{x_1, \ldots, x_k\}$，$x_i \in \mathbb{R}^3$这些点将用于近似能量函数中的 $\hat{M}$。

2. **属性采样：**

   对于每个 x_i，通过在其所属面上进行重心插值，计算其标量属性 $x_i \in \mathbb{R}^d$，从而得到对应于 X 的属性集合 $\underline{X}$：

   $\underline{X} = \{\underline{x_1}, \ldots, \underline{x_k}\}$，$\underline{x_i} \in \mathbb{R}^d$

3. **锐边采样：**

   额外采样一个由 k' 个点组成的集合 X'，这些点仅限于位于 $\hat{M}$ 曲面上的尖锐边：

   $X' = \{x'_1, \ldots, x'_{k'}\}$，$x'_i \in \mathbb{R}^3$

   尖锐边由以下部分组成：

   (a) 材料边界：面之间具有$f_i \neq f_j$的边。(b) 几何边界：仅连接一个面的边

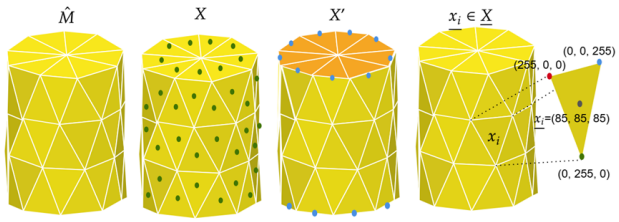

图18。展示原始网格和采样点集的示例

8.2.2. 代价函数定义

考虑经过若干次边折叠后的网格 $\hat{M}$ 。设 $M^{\uparrow}$ 为再次塌缩一条边之前的状态， M 为边折叠后的状态。此次塌缩的代价表述为：

$$\mathcal{E} = \underbrace{\left(E_{dist}(M) - E_{dist}\left(M^{\uparrow}\right)\right)}_{\Delta E_{dist}} + \underbrace{\left(E_{spring}(M) - E_{spring}\left(M^{\uparrow}\right)\right)}_{\Delta E_{spring}} + \underbrace{\left(E_{scalar}(M) - E_{scalar}\left(M^{\uparrow}\right)\right)}_{\Delta E_{scalar}} + D_{disc} \tag{4}$$

距离能量：

E_{dist} 通过惩罚每个采样点 $x_i \in X$ 与其在网格 M 上的投影 $p(x_i)$ 之间的距离，来衡量几何保真度。

$$E_{dist}(M) = \sum_{x_i \in X} \|x_i - p(x_i)\|^2 \tag{4a}$$

弹簧能量：

E_{spring} 通过将每条边视为一根“弹簧”，并将其长度的平方计入能量，对优化进行正则化。

$$E_{spring}(M) = \kappa \sum_{(v_a, v_b) \in \text{edges}(M)} \|v_a - v_b\|^2 \tag{4b}$$

这里，弹簧常数 κ 用于加权这一项。

标量属性能量：

E_{scalar} 与 E_{dist} 类似，但针对的是标量属性。它用于衡量每个采样点 x_i 的属性与其投影 $p(x_i)$ 的属性之间的差值平方：

$$E_{scalar}(M) = c_{scalar} \sum_{\underline{x_i} \in \underline{X}} \left\| \underline{x_i} - \underline{p}(x_i) \right\|^2 \tag{4c}$$

其中， c_{scalar} 用于加权这一项。

不连续性保持偏移：

D_{disc} 对塌缩会影响由 X' 跟踪的不连续性的锐边 e 施加惩罚：

$$D_{disc} = \text{numProject}(X', e)\, \|e\|^2 \tag{4d}$$

此处， $\text{numProject}(X', e)$ 表示 X' 投影到 e 上的点数。仅当塌缩改变不连续性连通性时，才会根据 [Hoppe 1996]中介绍的准则应用此项。若要完全禁止塌缩，请将 $D_{disc} = \infty$ 设为。

代价函数评估和最优顶点放置：

回想一下， $M^{\uparrow}$ 是当前边塌缩前的网格状态， M 是塌缩后的网格。新顶点被放置在位置 v ，其属性为 v。我们需要计算 v 和 v ，使代价 $\mathcal{E}$最小。

根据 公式 4，其中的各项 $\mathcal{E}$取决于 v 和 v ，如下所示:

- $E_{dist}(M)$ 和 $E_{spring}(M)$仅取决于 v。
- $E_{scalar}(M)$ 仅取决于 v。
- D_{disc} 以及 $M^{\uparrow}$ 的能量项独立于 v 或 v。因此，它们只影响代价 $\mathcal{E}$，而不影响优化。

因此，以下是针对给定边折叠计算 $\mathcal{E}$、 v 和 v 的步骤:

1. 在 v 上最小化 $\Delta E_{dist} + \Delta E_{spring}$。
2. 在v上最小化 ΔE_{scalar}。
3. 如果满足不连续性判据，则计算 D_{disc}。
4. 返回 $\mathcal{E} = \Delta E_{dist} + \Delta E_{spring} + \Delta E_{scalar} + D_{disc}$， 以及最优的 v 和 v。

8.2.3. 优化过程

最小化误差 $\mathcal{E}$ 比之前的误差函数更为复杂，并且需要采用分阶段优化过程。下面将详细介绍这一过程，而计算和优化误差所需的数据则通过图19中的一个示例展示。

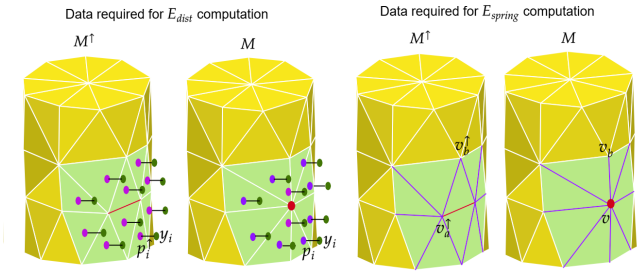

图19。展示所需数据的示例，用于 E_{dist} 和 E_{spring}

最小化 ΔE_{spring} over v

$$\begin{aligned}\Delta E_{spring} &= E_{spring}(M) - E_{spring}\left(M^{\uparrow}\right) \\ &= \kappa \left[\sum_{(v_a, v_b) \in \text{edges}(M)} \|v_a - v_b\|^2 - \sum_{\left(v_a^{\uparrow}, v_b^{\uparrow}\right) \in \text{edges}(M^{\uparrow})} \left\|v_a^{\uparrow} - v_b^{\uparrow}\right\|^2 \right]\end{aligned}$$

我们可以观察到，在ΔE_{spring}中， M and $M^{\uparrow}$ 中端点未发生变化的任何边都会相互抵消。由于差异仅出现在折叠边附近，因此只有两个边集会产生非零项：

1. N：在 v 处与新顶点关联的 M 中的边。
2. $N^{\uparrow}$：与折叠边关联的 $M^{\uparrow}$ 中的边。

因此，我们可以按如下方式简化 ΔE_{spring}：

$$\Delta E_{spring} = \kappa \left[\sum_{(v,v_b) \in N} \|v - v_b\|^2 - \sum_{\left(v_a^{\uparrow}, v_b^{\uparrow}\right) \in N^{\uparrow}} \left\|v_a^{\uparrow} - v_b^{\uparrow}\right\|^2 \right] \tag{5}$$

其中 v 会明确显示，因为 N 中的所有边都与它关联。
由于 $E_{spring}\left(M^{\uparrow}\right)$ 与 v无关，因此在优化过程中会忽略它，仅将其计入最终代价。

弹簧项 $E_{spring}(M)$,

$$E_{spring}(M) = \kappa \sum_{(v,v_b) \in N} \|v - v_b\|^2$$

是关于 v 的二次函数，可写为

$$\begin{aligned}
E_{spring}(M) &= \kappa \sum_{(v,v_b) \in N} \left(v^T v - 2 v_b^T v + v_b^T v_b \right) \\
&= \kappa \left(\sum_{(v,v_b) \in N} v^T v - \sum_{(v,v_b) \in N} 2 v_b^T v + \sum_{(v,v_b) \in N} v_b^T v_b \right) \\
&= \kappa \left(v^T (n_e I) v - 2 \left(\sum_{(v,v_b) \in N} v_b \right)^T v + \left(\sum_{(v,v_b) \in N} v_b^T v_b \right) \right) \quad \dots\ n_e \text{ is the number of edges in } N \\
&= \kappa \left(v^T H v + 2 c^T v + k \right)
\end{aligned}$$

对 v上的 ΔE_{dist} 进行最小化

$$\begin{aligned}
\Delta E_{dist} &= E_{dist}(M) - E_{dist}\left(M^{\uparrow}\right) \\
&= \sum_{x_i \in X} \left[\|x_i - p(x_i)\|^2 - \left\|x_i - p^{\uparrow}(x_i)\right\|^2 \right]
\end{aligned}$$

其中，同一采样点 x_i 在 M 中的投影为 $p(x_i)$，在 $M^{\uparrow}$中的投影为 $p^{\uparrow}(x_i)$。
由于 M和 $M^{\uparrow}$ 仅在正在塌缩的那条边附近存在差异，因此大多数点在两种状态下的投影都完全相同并相互抵消。因此，只有投影到该边邻域的点 $Y \subseteq X$ 会产生贡献：

$$\Delta E_{dist} = \sum_{y_i \in Y} \left[\|y_i - p(y_i)\|^2 - \left\|y_i - p^{\uparrow}(y_i)\right\|^2 \right] \tag{6}$$

我们将边的邻域定义为在 M 或 $M^{\uparrow}$中与该边任一端点相连的面的集合。

这种简化基于局部性假设：新顶点 v 位于折叠边附近，因此远处点在 X 中的投影保持不变。尽管允许 v 移得更远可能会降低代价，但这需要对所有 X重新计算。实践中，限制 v 并使用简化后的 ΔE_{dist} 效果很好。

由于 $E_{dist}\left(M^{\uparrow}\right)$ 独立于 v，因此优化过程中会忽略它，仅将其计入最终代价。

现在，尽管 $E_{dist}(M)$项，

$$E_{dist}(M) = \sum_{y_i \in Y} \|y_i - p(y_i)\|^2$$

看起来独立于 v，但每个投影 $p(y_i)$都取决于 v。

对于一个点 y，其投影 $p(y)$为

$$p(y) = \arg\min_{p} \|y - p\|^2$$

其中， p 位于 M 中的某个面上，该面由顶点 (v_a, v_b, v_c)构成。我们使用重心坐标 $\beta = (\beta_a, \beta_b, \beta_c)$ 来表示 p:

$$p = V\beta, \qquad V = \begin{bmatrix} v_a & v_b & v_c \end{bmatrix} \in \mathbb{R}^{3\times 3}, \qquad \beta \in [0,1]^3$$

我们可以定义一个函数 $\beta(y)$ ，用来表示任意点 y的投影重心坐标:

$$\beta(y) = \arg\min_{\beta} \|y - V\beta\|^2, \qquad p(y) = V\beta(y)$$

现在，由于 V 和 β 都与特定的面相关，我们将 V 扩展到包含 n 个顶点 $\{v_1, \ldots, v, \ldots, v_n\}$ 的完整网格 M ，并将任意给定的 β 转换为 n维向量，除 (v_a, v_b, v_c)之外，所有顶点对应的条目均置为零

$$\mathbf{V} = [v_1 \mid \ldots \mid v_n] \in \mathbb{R}^{3\times n}, \qquad \beta \in [0,1]^n$$

因此， $E_{dist}(M)$ 可以用 v （作为 $\mathbf{V}$的一部分）改写为:

$$E_{dist}(M) = \sum_{y_i \in Y} \|y_i - p(y_i)\|^2 \Longrightarrow \sum_{y_i \in Y} \|y_i - \mathbf{V}\beta(y_i)\|^2 \Longrightarrow \sum_{y_i \in Y} \min_{\beta} \|y_i - \mathbf{V}\beta\|^2$$

现在我们可以对 v上的 $E_{dist}(M)$ 进行最小化。由于计算它需要通过在 β 空间中进行内层最小化来投影每个 y_i，因此该问题是嵌套的：对 v 进行外层最小化，并对所有 $\beta(y_i)$进行内层最小化。

我们按照图20所示的方式，从 v 的初始猜测开始，交替进行以下步骤来迭代求解嵌套最小化问题：在固定 $\beta(y_i)$的情况下优化 v ，然后在固定 v的情况下更新$\beta(y_i)$ 。这一过程会持续到收敛，即每次迭代之间 v 以及每个$\beta(y_i)$的值变化不大为止。在实践中，少量迭代通常就足够了。

对于每个 $\beta(y_i)$ （固定 v ），进行的内层最小化称为投影子问题，即将 Y 中的所有点投影到 M上。蛮力方法是尝试将

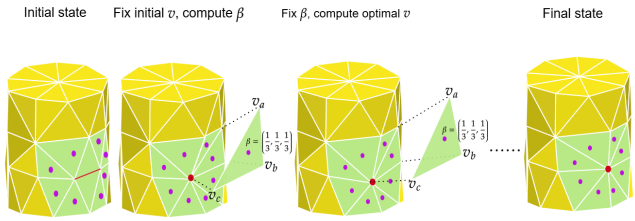

图20。 v 与每个 $\beta(y_i)$的交替优化

每个 y_i 投影到的每个面上并计算 $\beta(y_i)$，对应于距离最近的面。但是，[Hoppe 1996]为这种方法增加了两项加速措施:

1. 使用空间划分结构，以每个点 $O(1)$ 的时间复杂度查找候选面，在早期阶段或新区域发生边折叠后尤其有用。2. 如果将 y_i 投影到 $M^{\uparrow}$上，则将其在 M 上的投影限制为与前一个面相邻的面，从而利用局部性。

现在，通过利用该恒定性重写 $E_{dist}(M)$，即可求解外层最小化——即在保持所有 $\beta(y_i)$ 不变的情况下对 v 进行最小化。

$$
\begin{aligned}
E_{dist}(M) &= \sum_{y_i \in Y} \|y_i - \mathbf{V}\beta(y_i)\|^2 \\
&= \sum_{y_i \in Y} \left\| y_i - \begin{pmatrix} v_1 & \dots & v & \dots & v_n \end{pmatrix} \begin{pmatrix} \beta_1(y_i) & \dots & \beta_v(y_i) & \dots & \beta_n(y_i) \end{pmatrix}^T \right\|^2
\end{aligned}
$$

其中， $\beta_v(y_i)$ 表示 $\beta(y_i)$中的 $v's$ 分量。因此，我们有:

$$
\begin{aligned}
E_{dist}(M) &= \sum_{y_i \in Y} \left\| y_i - \left(\beta_v(y_i) v + \sum_{v_j \neq v} \beta_j(y_i) v_j \right) \right\|^2 \\
&= \sum_{y_i \in Y} \|z_i - \beta_v(y_i) v\|^2 \qquad \text{where } z_i = y_i - \sum_{v_j \neq v} \beta_j(y_i) v_j \\
&= \sum_{y_i \in Y} \left(\beta_v^2(y_i) v^T v - 2\beta_v(y_i) z_i^T v + z_i^T z_i \right) \\
&= v^T \left(\sum_{y_i \in Y} \beta_v^2(y_i) \right) I v + 2 \left(- \sum_{y_i \in Y} \beta_v(y_i) z_i \right)^T v + \left(\sum_{y_i \in Y} z_i^T z_i \right) \\
&= v^T H v + 2 c^T v + k
\end{aligned}
$$

这将得出与 $E_{spring}(M)$ 相同的二次误差形式，因此我们可以将两者的 $H\,c\,k$ 系数相加，并求解 v，从而对两者进行联合优化。

最小化 ΔE_{scalar} **over** v

成本函数 ΔE_{scalar} 与 E_{dist} 类似，并沿用其局部性假设，得到如下表达式

$$\begin{aligned}\Delta E_{scalar} &= E_{scalar}(M) - E_{scalar}\left(M^{\uparrow}\right) \\ &= c_{scalar} \sum_{\underline{y_i} \in \underline{Y}} \left[\left\| \underline{y_i} - p\left(\underline{y_i}\right) \right\|^2 - \left\| \underline{y_i} - p^{\uparrow}\left(\underline{y_i}\right) \right\|^2 \right]\end{aligned} \tag{7}$$

这里，$\underline{Y} \subseteq X$ 是样本属性向量的一个局部子集。我们将此邻域定义为与待塌缩边相邻的面上的所有样本点的集合，这与用于 E_{dist}的方法一致。

由于 $E_{scalar}\left(M^{\uparrow}\right)$ 独立于υ，因此在优化过程中会忽略它，只将其计入最终代价。

通过重用先前对 ΔE_{dist}进行优化时计算的重心坐标集合β 和 $\beta^{\uparrow}$，可以简化优化过程。

$p\left(\underline{y_i}\right)$ 是与投影点对应的属性向量。其值可通过以下方式计算：

$$p(y_i) = \mathbf{V}\beta\left(\underline{y_i}\right)$$

其中，$\mathbf{V}$ 是M的顶点属性堆叠矩阵，即 $\mathbf{V} = \left(\upsilon_1 \ldots \quad \upsilon \quad \ldots \quad \upsilon_m\right) \in \mathbb{R}^{d \times m}$。

因此，方程 7 可简化为：

$$\begin{aligned}E_{scalar}(M) &= \sum_{\underline{y_i} \in \underline{Y}} \left\| \underline{y_i} - p\left(\underline{y_i}\right) \right\|^2 \\ &= \sum_{\underline{y_i} \in \underline{Y}} \left\| \underline{y_i} - \underline{\mathbf{V}}\beta\left(\underline{y_i}\right) \right\|^2 \\ &= \sum_{\underline{y_i} \in \underline{Y}} \left\| \underline{y_i} - \left(\underline{\upsilon_1} \quad \ldots \quad \underline{\upsilon} \quad \ldots \quad \underline{\upsilon_n}\right)\left(\beta_1\left(\underline{y_i}\right) \quad \ldots \quad \beta_\upsilon\left(\underline{y_i}\right) \quad \ldots \quad \beta_n\left(\underline{y_i}\right)\right)^T \right\|^2\end{aligned}$$

其中，$\beta_\upsilon\left(\underline{y_i}\right)$表示 $\beta\left(\underline{y_i}\right)$中的 $\upsilon's$ 分量

$$\begin{aligned}E_{scalar}(M) &= \sum_{\underline{y_i} \in \underline{Y}} \left\| \underline{y_i} - \left(\beta_\upsilon\left(\underline{y_i}\right)\underline{\upsilon} + \sum_{\underline{\upsilon_j} \neq \underline{\upsilon}} \beta_j\left(\underline{y_i}\right)\underline{\upsilon_j} \right) \right\|^2 \\ &= \sum_{\underline{y_i} \in \underline{Y}} \left\| \underline{z_i} - \beta_\upsilon\left(\underline{y_i}\right)\underline{\upsilon} \right\|^2 \qquad \text{其中 } z_i = y_i - \sum_{\upsilon_j \neq \upsilon} \beta_j\left(\underline{y_i}\right)\underline{\upsilon_j} \\ &= \sum_{\underline{y_i} \in \underline{Y}} \left(\beta_\upsilon^2\left(\underline{y_i}\right)\underline{\upsilon}^T\underline{\upsilon} - 2\beta_\upsilon\left(\underline{y_i}\right)\underline{z_i}^T\underline{\upsilon} + \underline{z_i}^T\underline{z_i} \right) \\ &= \underline{\upsilon}^T \left(\sum_{\underline{y_i} \in Y} \beta_\upsilon^2\left(\underline{y_i}\right) \right) I\underline{\upsilon} + 2\left(-\sum_{\underline{y_i} \in Y} \beta_\upsilon\left(\underline{y_i}\right)\underline{z_i} \right)^T \underline{\upsilon} + \left(\sum_{\underline{y_i} \in Y} \underline{z_i^T z_i} \right) \\ &= \underline{\upsilon}^T H \underline{\upsilon} + 2c^T\underline{\upsilon} + k\end{aligned}$$

这可以像我们之前见过的其他约束一样求解。

9. 结论

本文通过深入研究四篇对该算法提出不同变体的重要论文，详细探讨了使用边塌缩进行网格简化： [Gar- land 和 Heckbert 1997；Lindstrom and Turk1998；Garland 和 Heckbert 1998；Hoppe1996]。

我们首先讨论了网格简化的基础知识，以及用于实现这一目标的不同类别算法。随后，我们重点介绍了边折叠和通常用于实现它的半边数据结构。接着，我们概述了通过边折叠进行简化所采用的一般算法，其中包括需要考虑的重要边界情况。然后，我们通过多种度量对候选顶点位置所引入的误差计算过程进行了详尽分析，并讨论了如何汇总相关约束，构成一个可求解的线性方程组，从而为给定的边折叠求得最终的最优顶点放置。在此过程中，我们还处理了执行网格简化时的其他重要问题，例如边界边以及顶点/面属性的处理。

我们相信，这项工作能够帮助对几何处理和网格简化感兴趣的人深入理解这些强大的算法和度量，将其应用于自己的使用场景，并启发该领域的进一步研究。

参考文献

Chen, Y.-C., Kim, V., Aigerman, N. 和 Jacobson, A. 神经渐进网格。载于 ACM SIGGRAPH 2023 会议论文集，第 1–9 页，2023 年。4 Cignoni, P., Montani, C. 和 Scopigno, R. 网格简化算法比较。《计算机与图形学》，22(1):37–54，1998 年。3 Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F. 和 Wright, W. 简化包络。载于第 23 届计算机图形学与交互技术年会论文集，第 119–128 页，1996年。4 DeHaemer Jr, M. J. 和 Zyda, M. J. 多边形近似渲染对象的简化。《计算机与图形学》，15(2):175–184，1991 年。3 Garland, M. 和 Heckbert, P. S. 使用二次误差度量的曲面简化。载于第 24 届计算机图形学与交互技术年会论文集，SIGGRAPH ’97，第 209–216 页，美国，1997 年。ACM Press/Addison-Wesley 出版公司。4, 9, 41 Garland, M. 和 Heckbert, P. S. 使用二次误差度量简化带颜色和纹理的曲面。载于 Visualization ’98 会议论文集，VIS ’98，第 263–269 页，美国华盛顿特区，1998 年。IEEE 计算机协会出版社。11, 12, 13, 31, 41 He, T., Hong, L., Kaufman, A., Varshney, A. 和 Wang, S. 基于体素的对象简化。载于 Visualization’95 论文集，第 296–303 页。IEEE，1995 年。4

He, T., Hong, L., Varshney, A. 和 Wang, S. W. 受控拓扑简化。《IEEE 可视化与计算机图形学汇刊》，2(2):171–184，1996年。4 Hinker, P. 和 Hansen, C. 几何优化。载于 Visualization’93 论文集，第 189–195 页。IEEE，1993 年。3 Hoppe, H. 渐进网格。载于第 23 届计算机图形学与交互技术年会论文集，SIGGRAPH ’96，第 99–108 页，美国纽约州纽约市，1996年。计算机协会。33, 35, 39, 41 Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. 和 Stuetzle, W. 网格优化。载于第 20 届计算机图形学与交互技术年会论文集，第 19–26 页，1993 年。4, 7 Lindstrom, P. 和 Turk, G. 快速且节省内存的多边形简化。载于 Visualization ’98 论文集（编号 98CB36276），第 279–286 页，1998 年。网址：https: // doi.org/10.1109/VISUAL.1998.745314。4, 14, 19, 23, 41 Low, K.-L. 和 Tan, T.-S. 使用顶点聚类进行模型简化。载于 1997 年交互式三维图形研讨会论文集，第 75 页起，1997 年。3 McGuire, M. 半边数据结构。网站：http://www. flipcode. com/articles/ar- ticle halfedgepf. shtml，2000 年。5 Park, J. J., Florence, P., Straub, J., Newcombe, R. 和 Lovegrove, S. DeepSDF：用于形状表示的连续有符号距离函数学习。载于 IEEE/CVF计算机视觉与模式识别会议论文集，第 165–174 页，2019 年。4 Potamias, R. A., Ploumpis, S. 和 Zafeiriou, S. 神经网格简化。载于 IEEE/CVF计算机视觉与模式识别会议论文集，第 18583–18592 页，2022 年。4 Rossignac, J. Edgebreaker：三角网格的连通性压缩。《IEEE 可视化与计算机图形学汇刊》，5(1):47–61，2002 年。5 Rossignac, J. 和 Borrel, P. 用于渲染复杂场景的多分辨率三维近似，第 455–465 页。施普林格，1993 年。3 Schroeder, W. J., Zarge, J. A. 和 Lorensen, W. E. 三角网格抽取。载于第 19 届计算机图形学与交互技术年会论文集，第 65–70 页，1992 年。4 Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M. 和 Fidler, S. 神经几何细节层次：使用隐式三维形状进行实时渲染。载于 IEEE/CVF计算机视觉与模式识别会议论文集，第 11358–11367 页，2021 年。4 Turk, G. 多边形曲面的重新铺设。载于第 19 届计算机图形学与交互技术年会论文集，第 55–64 页，1992 年。4

附录

说明：平面边界和非平面边界在边界保持误差方面得出的公式相同。

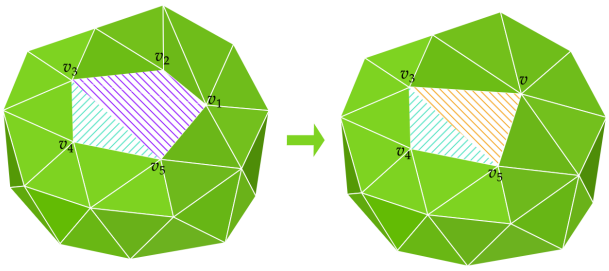

图 21。边界保持——非平面边界情况

当边 (v_1, v_2) 被顶点 v 替换时，如 图 21所示， $Area(v_1, v_2, v_3, v_5)$发生变化，但 $Area(v_5, v_3, v_4)$ 保持不变。因此，这种情况下的面积变化 $\mathcal{E}'$为：

$$\begin{aligned}
\mathcal{E}' &= Area(v_1, v_2, v_3, v_5) - Area(v, v_3, v_5) \\
&= Area(v_1, v_2, v_3) + Area(v_1, v_3, v_5) - Area(v, v_3, v_5) \\
&= \frac{1}{2}\begin{pmatrix} (v_1 \times v_2) + (v_2 \times v_3) + \cancel{(v_3 \times v_1)} + \\ \cancel{(v_1 \times v_3)} + \cancel{(v_3 \times v_5)} + (v_5 \times v_1) + \\ (v_3 \times v) + \cancel{(v_5 \times v_3)} + (v \times v_5) \end{pmatrix} \\
&= \frac{1}{2}\left((v_1 \times v_2) + (v_2 \times v_3) + (v_5 \times v_1) + (v_3 \times v) + (v \times v_5)\right)
\end{aligned}$$

根据平面边界面积变化的计算，我们知道，

$$\begin{aligned}
\mathcal{E} &= \frac{1}{2}\begin{pmatrix} \cancel{(v \times v_1)} + (v_1 \times v_2) + \cancel{(v_2 \times v)} + \\ \cancel{(v \times v_2)} + (v_2 \times v_3) + (v_3 \times v) + \\ (v \times v_5) + (v_5 \times v_1) + \cancel{(v_1 \times v)} \end{pmatrix} \\
&= \frac{1}{2}\left((v_1 \times v_2) + (v_2 \times v_3) + (v_3 \times v) + (v \times v_5) + (v_5 \times v_1)\right)
\end{aligned}$$

因此我们证明了 $\mathcal{E} = \mathcal{E}'$，这意味着平面边界和非平面边界的面积变化计算相同。

说明：在体积保持中，四面体的体积由 $V =$ 给出

$$\frac{1}{6}\begin{vmatrix} v_x & v_{1x}^t & v_{2x}^t & v_{3x}^t \\ v_y & v_{1y}^t & v_{2y}^t & v_{3y}^t \\ v_z & v_{1z}^t & v_{2z}^t & v_{3z}^t \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

为了简化量hV，我们应用列变换 $C_2 = C_2 - C_1$、 $C_3 = C_3 - C_1$ 和 $C_4 = C_4 - C_1$。由此得到：

$$V = \frac{1}{6}\begin{vmatrix} v_x & v_{1x}^t - v_x & v_{2x}^t - v_x & v_{3x}^t - v_x \\ v_y & v_{1y}^t - v_y & v_{2y}^t - v_y & v_{3y}^t - v_y \\ v_z & v_{1z}^t - v_z & v_{2z}^t - v_z & v_{3z}^t - v_z \\ 1 & 0 & 0 & 0 \end{vmatrix}$$

沿最后一行展开此行列式，可将其化简为一个 3×3 行列式：

$$V = -\frac{1}{6}\begin{vmatrix} v_{1x}^t - v_x & v_{2x}^t - v_x & v_{3x}^t - v_x \\ v_{1y}^t - v_y & v_{2y}^t - v_y & v_{3y}^t - v_y \\ v_{1z}^t - v_z & v_{2z}^t - v_z & v_{3z}^t - v_z \end{vmatrix}$$

忽略前面的常数后，我们可以注意到，具有基向量 C_1、 C_2 和 C_3 的矩阵的行列式给出了它们所张成的平行六面体的有向体积，如图 22所示。

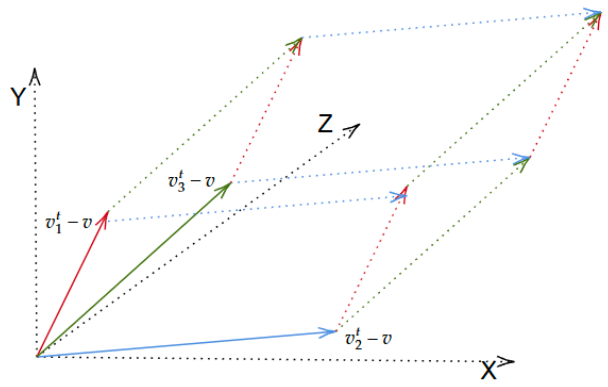

图 2。 2 四面体平行六面体体积示意 图

此外，我们可以从几何上证明，平行六面体包含六个体积相等的四面体——见图 23。

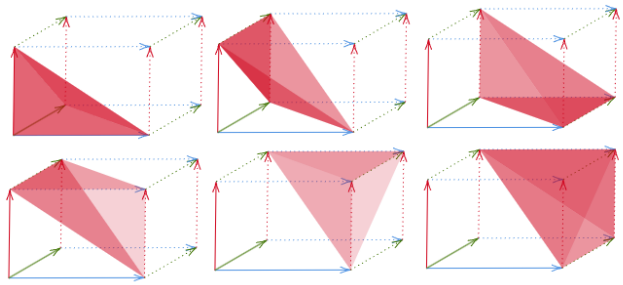

图 23. 一个平行六面体中的 6 个全等四面体

因此，要得到单个四面体的体积，我们将平行六面体的体积除以 6。

辅助函数

```
命名空间 util
{
  模板 <typename T>
  向量<const T*> 求并集(const 向量<const T*>& vector1 , 常量
      向量<常量 T*>& 向量2)
  {
    集合<常量 T*> 唯一_指针;

    唯一_指针.插入(向量1.开始(), 向量1.结束());
    唯一_指针.插入(向量2.开始(), 三维向量2.结束());

    返回 向量<常量 T*>(唯一_指针.开始(), 唯一_指针
        。结束());
  }

  布尔值是边界边(常量IMesh* 网格,常量IEdge* 边)
  {
    返回 网格->获取连接的面(边).大小() < 2;
  }

  三维向量 计算法向量(常量IFace* 面)
  {
    自动 面 = 面->获取顶点();
    三维向量 法向量 = 叉积(
      面[1].获取位置() - 面[0].获取位置() ,
      面[2].获取位置() - 面[0].获取位置()
    );
    返回 归一化(法向量);
  }

  三阶矩阵 构造反对称矩阵(常量 三维向量& v)
  {
    三阶矩阵斜_对称_矩阵 = 三阶矩阵(0);

    斜_对称_矩阵[0][1] = -v[2]; 斜_对称_矩阵[0][2] = v[1];
    斜_对称_矩阵[1][0] = v[2]; 斜_对称_矩阵[1][2] =-v[0];
    斜_对称_矩阵[2][0] = -v[1]; 斜_对称_矩阵[2][1] = v[0];
  }
}
```

作者联系信息

Purva Kulkarni 独立研究员 1400 Main St. 卡农斯堡，宾夕法尼亚州 1 5317purvaskulkarni14 @gmail.com

Aravind Shankara Narayanan 独立研究员 9805 Jake Lane 圣迭戈，加利福尼亚州 92126aravind.rssn@gmail.com