计算几何大实验——基于 QEM 的网格简化方法

姓名: 郭晟宇　　学号：2024312356
姓名: 王麒杰　　学号：2024213659
姓名: 胡昌宇　　学号：2024311574

1　引言

三维网格模型是计算机图形学中表示三维物体表面几何形状的主要方式，通常由大量的顶点、边和面构成。在许多应用场景下，例如虚拟现实、计算机游戏、科学计算可视化等，高精度的模型虽然能够提供丰富的细节，但其庞大的数据量往往会导致渲染效率低下，甚至超出硬件的处理能力。因此，研究高效的网格简化算法，在保证视觉效果的同时有效降低模型复杂度，具有重要的理论意义和应用价值。

2　网格简化算法分类

为了在保留关键几何特征的同时减少网格模型的复杂度，学术界和工业界已经发展出两大类主流技术路线：**网格削减 (Mesh Reduction)** 和 **网格重建 (Re-meshing)**。前者在原始网格拓扑结构上进行迭代修改，而后者则生成一个全新的、拓扑独立的网格来逼近原始形状。选择哪种路线通常取决于最终应用对简化率和模型保真度的具体要求。

2.1　网格削减 (Mesh Reduction)

网格削减技术通过一系列渐进的、局部的简化操作来直接减少原始模型中的顶点和面片数量。这类方法的核心在于其迭代的本质：每一步都试图以最小的代价移除一个或多个几何元素。由于它们保留了原始模型的大部分顶点数据和连接性，因此非常适合于中等程度的简化，并能在简化过程中较好地控制误差。其主要操作类型包括：

2.1.1　顶点抽取 (Vertex Decimation)

这是最早的简化方法之一。其基本流程是：

1. 从当前网格的顶点集合 V 中选择一个顶点 **v** 进行删除。
2. 移除顶点 **v** 以及所有与它相邻的三角面片。
3. 此时，围绕 **v** 的邻接顶点会形成一个“空洞”（一个多边形）。
4. 对这个空洞进行重新三角化，将其填充，从而完成一次简化。

该方法的关键挑战在于第一步，即如何选择要删除的顶点。选择标准通常基于局部几何特征，例如，优先删除那些位于平坦区域、曲率较低的顶点，以保留模型的高曲率特征（如棱角）。

2.1.2 边收缩 (Edge Contraction)

边收缩是目前最流行、效果最均衡的网格削减操作，也是后续 QEM 算法的基础。该操作选择网格中的一条边 $(\mathbf{v}_1, \mathbf{v}_2)$，并将它的两个端点合并成一个单独的新顶点 $\mathbf{v}'$。所有原来连接到 $\mathbf{v}_1$ 或 $\mathbf{v}_2$ 的边现在都将连接到 $\mathbf{v}'$，这个过程中会移除两个三角面片。

这个看似简单的操作引出了两个核心问题，不同的算法正是在这两个问题上给出了不同的答案：

1. **代价评估 (Cost Evaluation):** 在众多条边中，应该优先收缩哪一条？为了获得高质量的简化结果，算法需要一个精确的度量标准来评估每次边收缩操作所引入的几何误差。一个好的代价函数会优先收缩那些对模型整体外观影响最小的边。许多算法，包括经典的二次误差度量（QEM）[3]，都致力于设计高效且准确的误差度量函数。

2. **新顶点放置 (New Vertex Placement):** 合并后的新顶点 $\mathbf{v}'$ 应该放在哪里？最简单的策略是将其放在原有两个顶点之一或它们的中点。然而，为了更好地逼近原始表面，更先进的策略会计算一个能使局部几何误差最小化的“最优点”位置。

2.1.3 点对收缩 (Pair Contraction)

点对收缩是边收缩的一种泛化。它不再局限于收缩由边直接连接的两个顶点，而是允许收缩网格上**任意一对**顶点 $(\mathbf{v}_i, \mathbf{v}_j)$，无论它们之间是否相邻。

这种方法提供了更大的灵活性，例如，它可以用于合并两个不相连的物体部分，或者快速地封闭模型上的狭长缝隙和孔洞。然而，这种灵活性也带来了更高的风险：不受约束的点对收缩可能会导致剧烈的拓扑变化（例如，产生非流形结构），因此需要更复杂的约束和代价评估机制来避免产生视觉上不可接受的结果。

2.1.4 顶点聚类 (Vertex Clustering)

顶点聚类提供了一种完全不同的、非迭代的削减思路。该方法首先将模型的包围盒划分成一个三维空间网格（Grid）。然后，所有落入同一个网格单元（Cell）内的顶点将被合并成一个唯一的代表性顶点。这个代表性顶点的坐标通常是单元内所有顶点坐标的平均值或加权平均值。

这种方法的优点是速度极快且易于实现，其时间复杂度与顶点数量成线性关系。但缺点也同样明显：它对拓扑结构的控制非常弱，很容易将邻近但无关联的表面部分错误地缝合在一起，并且简化结果的质量通常不高。

2.2 网格重建 (Re-meshing)

与在原始网格上“修修补补”的削减方法不同，网格重建旨在生成一个全新的、结构更优的网格来逼近原始模型。当需要进行极端简化，生成“low-poly”艺术风格或性能要求极高

的 LOD（Level of Detail）模型时，这类方法往往能提供更好的结果，因为它们不受限于原始网格的拓扑结构。其主要技术方向包括：

- **基于体素化的重建 (Voxelization Based Re-meshing):** 将模型转化为规整的体素表示，然后提取其表面生成网格 [7]。
- **基元拟合 (Primitive Fitting):** 使用平面、长方体等简单几何基元组合来逼近物体形状 [8]。
- **视觉驱动的方法 (Visual-driven Approaches):** 利用可微分渲染等技术，通过最小化渲染图像的差异来反向优化低多边形网格的形状 [4]。
- **基于学习的方法 (Learning-based Methods):** 使用深度神经网络直接从高模、点云甚至图像中生成低多边形网格 [1]。

3 QEM 网格简化算法介绍

3.1 基本思想

QEM（Quadric Error Metrics）是一种高效且广泛使用的三维网格简化算法，其核心思想是通过一系列 **边折叠（Edge Collapse）**操作逐步减少网格的顶点和面片数。在简化过程中，QEM 使用一种称为 **误差二次矩阵（Quadric Matrix）**的方法，来度量每次折叠操作带来的几何误差，从而指导简化过程，使最终简化后的网格尽量保留原始几何特征。

3.2 几何误差度量：误差二次矩阵

为了评估顶点合并对几何形状的影响，QEM 算法引入了误差二次矩阵（Quadric Error Matrix）来量化一个点偏离原始表面（多个三角面片构成）的程度。

设一个三角面片位于空间中，其平面方程为：

$$ax + by + cz + d = 0$$

将其改写为齐次形式：

$$\mathbf{n}^T\mathbf{p} = 0, \quad \text{其中}\mathbf{n} = [a, b, c, d]^T, \quad \mathbf{p} = [x, y, z, 1]^T$$

点 $\mathbf{p}$ 到该平面的有符号距离为 $\mathbf{n}^T\mathbf{p}$，其平方即为误差：

$$\text{error}(\mathbf{p}) = (\mathbf{n}^T\mathbf{p})^2 = \mathbf{p}^T(\mathbf{n}\mathbf{n}^T)\mathbf{p}$$

定义：

$$Q_f = \mathbf{n}\mathbf{n}^T \in \mathbb{R}^{4\times4}$$

称为面片 f 的 Quadric 矩阵。对于一个顶点 v，其属于多个三角面片，定义该点的误差二次矩阵为所有相关面的 Quadric 的和：

$$Q_v = \sum_{f\in\text{adj}(v)} Q_f$$

该矩阵可唯一确定点 v 的误差函数，使得任意新位置 $\mathbf{p}$ 的几何误差可由：

$$\text{error}(\mathbf{p}) = \mathbf{p}^T Q_v \mathbf{p}$$

直接计算得到。

该度量同时考虑了法向量方向（即面片的几何信息）与顶点在该法向量上的偏移，因此能够有效度量简化过程中对表面形状的破坏程度。

3.3 边折叠与最优位置计算

网格简化的基本操作是 **边折叠（Edge Collapse）**，即将边 (v_1, v_2) 收缩为一个新点 v_{new}。关键问题有两个：

1. **如何选择折叠边**：我们需要为所有合法边计算其折叠误差代价 $\text{cost}(v_{new})$，从中选择误差最小的边优先折叠。
2. **如何确定折叠后新点的位置**：需在三维空间中找到一个最优的新点 v_{new}，使得折叠后误差最小。

两个顶点合并后的误差矩阵为：

$$Q = Q_{v_1} + Q_{v_2}$$

定义新点 $\mathbf{v}_{new} = [x, y, z, 1]^T$，其误差为：

$$\text{cost}(\mathbf{v}_{new}) = \mathbf{v}_{new}^T Q \mathbf{v}_{new}$$

为使代价最小化，我们令该函数对 x, y, z 的偏导为零，构造如下最小二乘问题：

$$\min_{\mathbf{v}\in\mathbb{R}^3} \quad \begin{bmatrix} \mathbf{v} \\ 1 \end{bmatrix}^T Q \begin{bmatrix} \mathbf{v} \\ 1 \end{bmatrix}$$

记 Q 为：

$$Q = \begin{bmatrix} A & \mathbf{b} \\ \mathbf{b}^T & c \end{bmatrix}, \quad A \in \mathbb{R}^{3\times 3}, \mathbf{b} \in \mathbb{R}^3, c \in \mathbb{R}$$

最小化目标函数可转化为求解线性系统：

$$\frac{\partial}{\partial \mathbf{v}} \left(\mathbf{v}^T A \mathbf{v} + 2\mathbf{b}^T \mathbf{v} + c \right) = 2A\mathbf{v} + 2\mathbf{b} = 0 \Rightarrow A\mathbf{v} = -\mathbf{b}$$

若矩阵 A 可逆，则有：

$$\mathbf{v}_{new} = -A^{-1}\mathbf{b}$$

若 A 不可逆（退化情况如平面退化），可使用备选方案：

- 中点 $\dfrac{v_1 + v_2}{2}$
- 或遍历 v_1, v_2 与中点三者代入误差函数选最小值

最终，对于所有边 (v_1, v_2)，计算其最优折叠点和对应的 cost 值，使用优先队列维护边集，逐步折叠最小代价边。

4 进阶算法实现

在实现了基础的二次误差度量（QEM）算法后，为了应对不同应用场景下的特定需求，我们探索并实现了三种进阶的网格简化算法。这些算法均以经典的边收缩框架为基础，但通过引入不同的代价函数（Cost Function）来优化简化过程，旨在更好地保持模型的特定属性。我们的实现思路遵循一个从局部到全局的递进过程：首先通过引入局部约束来改良 QEM，分别实现了结合体积保持的算法和结合结构感知的算法；最后，为了从更宏观的尺度上保持模型特征，我们实现了一种基于谱分析的全局特征保持算法。

4.1 结合体积保持的 QEM 算法

标准的 QEM 算法仅关注最小化几何误差，这在简化率较高时可能导致模型产生不自然的体积收缩，尤其对于封闭模型，简化后的结果可能“干瘪”失真。为了解决这一问题，我们实现了 Lindstrom 和 Turk 提出的方法 [6]，其核心思想是在 QEM 的代价函数中额外引入**体积保持**和**边界保持**的惩罚项。

4.1.1 体积保持代价

在我们的实现（lindstrom_turk.cpp）中，体积保持通过函数 compute_volume_constraint 实现。其原理是，对于待收缩边 $(\mathbf{v}_1, \mathbf{v}_2)$，我们考虑所有以 $\mathbf{v}_1$ 或 $\mathbf{v}_2$ 为顶点的三角面片。每个这样的三角面片与模型内部的某个任意点（通常是原点）可以构成一个四面体。这条边收缩前后，这些相关四面体的有向体积会发生变化。我们计算这个体积变化的绝对值之和，作为体积保持的代价。这可以有效惩罚那些会导致模型总体积剧烈变化的收缩操作。

4.1.2 边界保持代价

为了避免简化过程中模型边界被破坏，我们通过函数 is_boundary_edge 判断一条边是否位于网格的边界上（即该边仅被一个面片共享）。如果是边界边，compute_boundary_constraint 会计算新顶点位置与原始边中点的距离，以此作为边界惩罚项，鼓励新顶点保持在原始边界上。

4.1.3 混合代价函数

最终的边收缩代价 compute_lindstrom_turk_cost 是一个加权和：

$$\text{Cost} = C_{\text{quadric}} + w_{\text{volume}} \cdot C_{\text{volume}} + w_{\text{boundary}} \cdot C_{\text{boundary}}$$

其中 C_{quadric} 是标准的二次误差，而 w_{volume} 和 w_{boundary} 是可调节的权重参数，用于控制体积和边界保持的强度。通过调整这两个权重，我们可以在几何精度和体积保持之间取得平衡。

4.2 结合结构感知的 QEM 算法

对于建筑、机械零件等包含大量平面和清晰棱线的模型，标准的简化算法常常会破坏这些重要结构。为此，我们实现了一种结构感知的简化算法 [9]（structure_aware.cpp），其核心在于识别并保护这些结构特征。

4.2.1 平面代理 (Planar Proxies)

算法首先通过一个预处理步骤 initialize_proxies 来识别模型表面的平面区域。在我们的实现中，我们采用了一种简化的方式来演示其概念，即每个三角面片初始时都构成一个独立的“平面代理”。在更完整的实现中，这里通常会使用法线相似度等准则进行区域生长，将近似共面的面片聚合成一个更大的平面代理。

4.2.2 结构感知的代价函数

代价函数 compute_structure_edge_cost 被设计为能够感知和保护这些代理结构：

1. **修改二次误差矩阵：**在计算顶点的初始二次误差矩阵时，我们引入了其所在代理的平面方程，使得顶点的移动代价不仅取决于局部几何，还受到其所属宏观平面结构的约束。
2. **边界二次项：**对于处于两个不同平面代理交界处的边（即模型的棱线），我们通过 compute_boundary_quadric 额外增加一个惩罚项，该惩罚项旨在惩罚任何会使该棱线“变钝”的收缩操作。
3. **拓扑约束：**在简化过程中，算法会检查一次收缩是否会导致两个不相邻的平面代理区域融合，或者是否会破坏一个角点（由三个或以上平面代理交汇的顶点）。通过赋予这类操作极高的代价，可以有效保护模型的宏观结构。

通过这种方式，算法能够以更高的优先级保留模型的平面和尖锐特征，生成结果在视觉上更加忠实于原始设计意图。

4.3 基于谱分析的全局特征保持算法

上述两种方法都是基于局部信息对 QEM 进行改良。为了从全局视角保持模型的整体形态，我们进一步实现了一种基于谱分析的网格简化算法 [5]（spectral2 .cpp）。该方法利用图拉普拉斯算子的谱（即特征值和特征向量）来捕捉和度量网格的全局形状特征。

4.3.1 网格拉普拉斯算子与谱

网格可以被看作一个图，其顶点和边构成了图的节点和连接。网格的拉普拉斯算子 L 是一个矩阵，它描述了顶点之间的连接关系。在我们的实现中，build_laplacian_matrix 构建了在几何处理中常用的**余切拉普拉斯算子**。

拉普拉斯算子的特征向量构成了网格上的一组“基函数”，其中与较小特征值（低频信号）对应的特征向量描述了模型的全局、平滑的宏观形状。保留这些低频特征，就等同于保留了模型的整体外观。

4.3.2 谱代价函数

算法的核心在于 compute_spectral_cost 函数，它定义了一个全新的、基于谱的代价。

1. **顶点重要性：**首先，通过 compute_eigenvectors 计算拉普拉斯矩阵的少量低频特征向量。然后，compute_vertex_importance 根据每个顶点在这些低频特征向量中的分量大

小，计算出一个“顶点重要性”值。一个顶点如果在多个全局形状基函数中都有很大贡献，则它被认为是重要的。

2. **谱距离：**对于待收缩的边 $(\mathbf{v}_1, \mathbf{v}_2)$，我们计算它在谱空间中的距离 compute_spectral_distance，即两个顶点在所有已计算的特征向量上分量的差值平方和。这个距离衡量了两个顶点在全局形状意义上的差异。

3. **混合代价：**最终的代价是几何代价与谱代价的加权和：

$$\text{Cost} = w_{\text{geom}} \cdot C_{\text{geom}} + w_{\text{spec}} \cdot C_{\text{spec}}$$

其中谱代价 C_{spec} 由谱距离和顶点重要性共同决定，这使得算法在收缩边时，会优先处理那些在全局形状上不重要且彼此相似的顶点对。

这种全局方法不依赖于局部的平面或体积假设，而是通过一种更抽象的数学方式来理解和保持模型的宏观形态，为高质量的网格简化提供了另一条有效的技术路径。

5 算法可视化

本次实验的可视化部分使用了 pyQt5 作为前端可视化，后端则使用 C++ 编写，编译成.so/.dll 库文件并使用 Pybind 提供 Python 部分的接口被前端调用。可视化的窗口效果如图 1所示

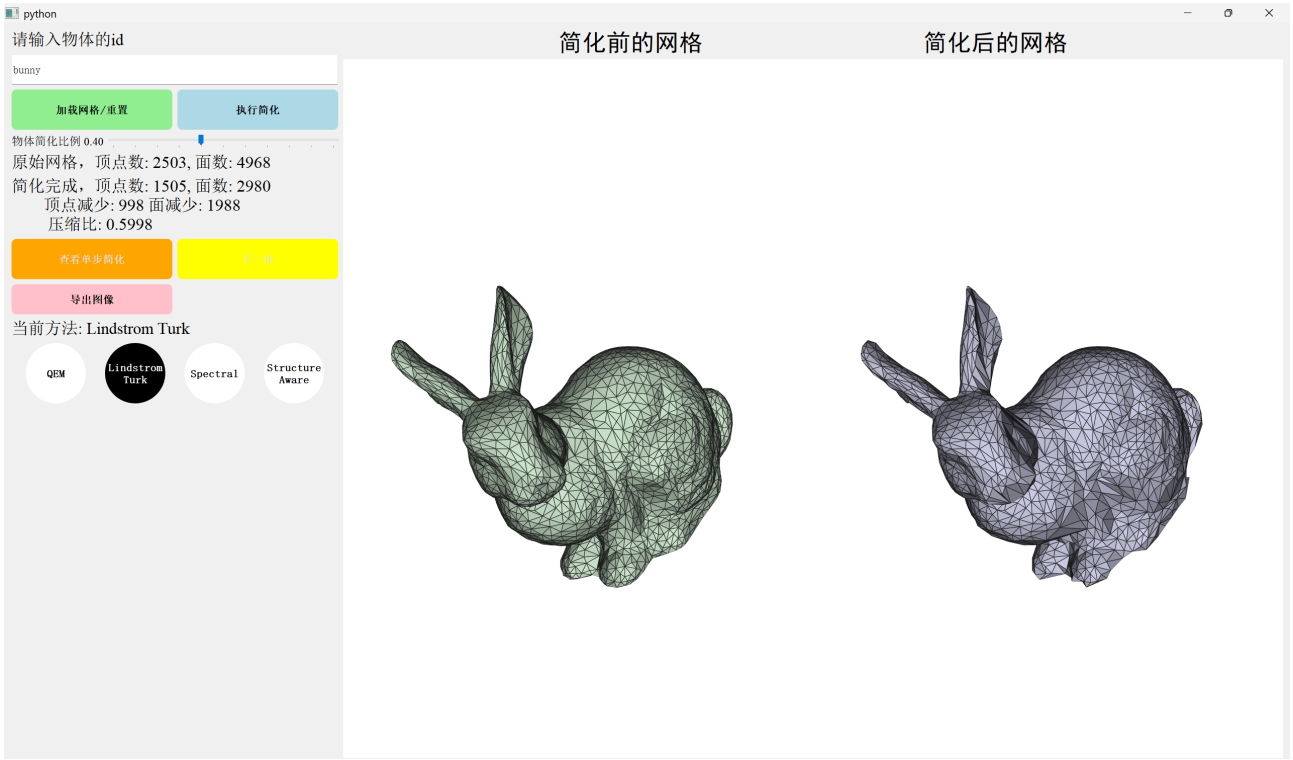

Figure 1: 可视化界面

5.1 操作界面

所有的按钮均使用了类 QPushButton，并自定义为不同的样式以及不同的位置，每个按钮通过信号槽机制连接到对应的槽函数上，比如“执行简化”按钮对应的槽函数即为调用后端的简化算法，返回简化后的网格并使用不同的颜色返回。下方的算法选择按钮，则链接了更换算法的槽函数，具体做法为重新加载算法对应的库文件，初始化一个新的后端。而简化比率则使用了 QSlider，以控制算法简化掉原本的多少比例面片后终止。

5.2 显示界面

所有的文本提示由 QLabel 显示，并在不同的状态下显示不同的文字，例如在加载网格后显示网格的面片顶点数，并在执行简化后显示简化后的面片定点数。右侧的渲染窗口则使用 QVTKRenderWindowInteractor，在其中含有两个子窗口各含有一个 vtkRenderer，用于实时渲染 mesh，同时设置二者使用同一个相机位姿，即拖动一侧的网格，另一侧的也会旋转至对应角度，以便于观察。

5.3 单步简化

实验内容与步骤

为了方便理解简化的流程，这里加入了“单步简化”机制，并通过可视化手段展示关键步骤。这里每次操作只合并一组顶点，并更新简化后的网格，同时配合相机动画和顶点动画。具体的步骤如下：

1. 简化操作执行（simplify_single_step）

- 调用 `SimplifyStep()` 得到新的顶点、面片及当前被简化的一组三个顶点；
- 确定受影响的面片，并对其上色高亮；
- 使用 `render_mesh` 分别渲染简化前后的网格；
- 更新当前网格状态，供下一步继续使用。

2. 相机动画聚焦（animate_camera_to）

- 获取当前正在被简化的面片，计算其中心和法向量；
- 设置相机飞行动画，从当前位置移动至面片正上方；
- 动画结束后自动进入顶点合并阶段。

3. 顶点动画合并（animate_vertex_move）

- 使用插值方式，将一个顶点平滑移动至另一个顶点位置；
- 每帧更新顶点坐标并刷新窗口，形成动态效果；

- 动画结束后显示简化结果（顶点数、面数）。

示意图如图 2，红色部分表示需要合并的边所连的两个三角面片，最后右侧图中的红色部分将会完全消失。

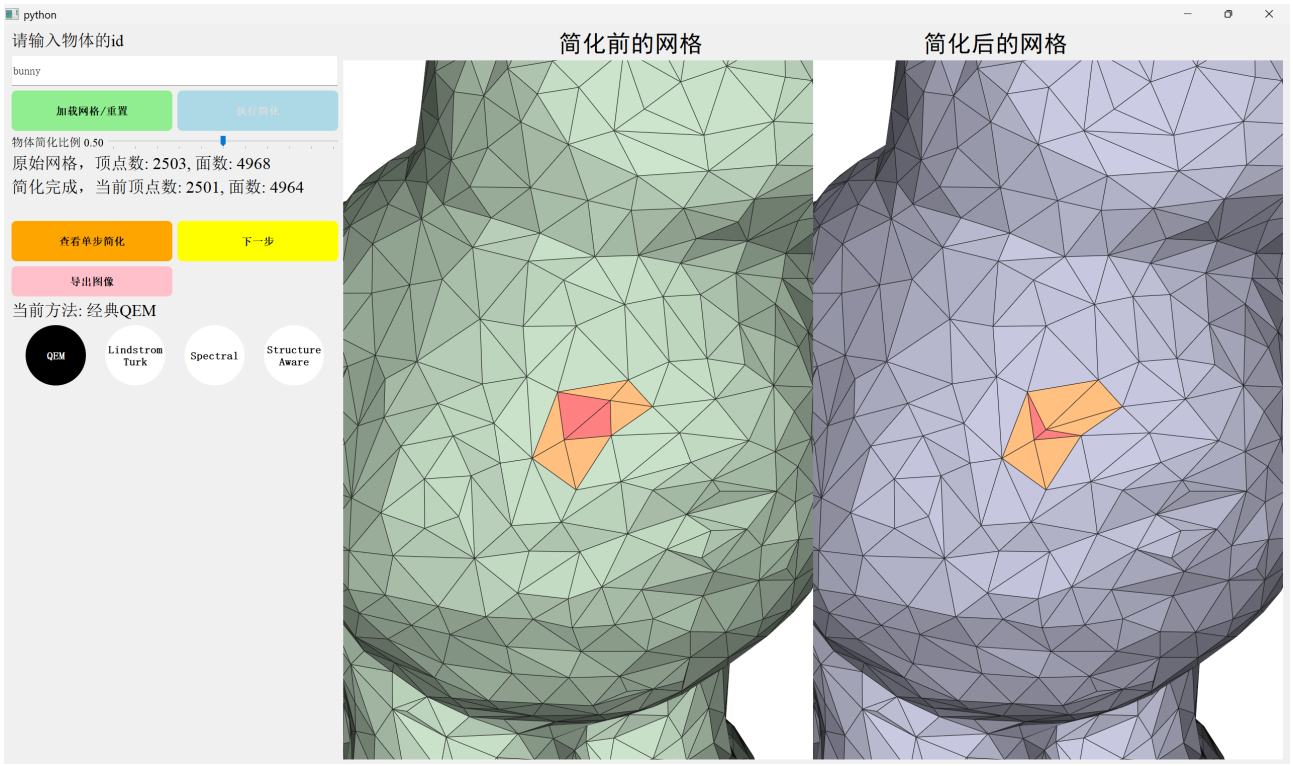

Figure 2: 正在进行简化的网格

6 实验

本次实验，对比了本次实验的原始 QEM 方法，以及其他三种方法，包括 Lindstrom-Turk，Spectral，Structure-Aware。实验选取了数据集 Objaverse [2]，这是一个新的三维数据集，从包含了 800K 个从建模网站 Sketch fab 上获取的物体。我们在其中选取出了一个子数据集（某人科研项目中采样的一个测试集），其中包含 550 个各种各样的物体，面片数量由 12 到 50000+。

6.1 数据集处理

由于 Objaverse 数据集源自建模网站，而为了纹理绘制的方便，通常的网格会被分为多个 shard，其中 shard 之间的顶点并非同一顶点。这样的非水密网格，在网格简化的过程中，会被大量减去 shard 边缘的面片，会导致视觉上不合规范的结果。因此我们需要对其进行水密化处理。

具体过程为，将物体包围盒所在的空间分为 $10^5 * 10^5 * 10^5$ 的体素，并将所有处于同一体素中的顶点合并为同一顶点，之后处理对应的边连接关系。经过这一处理，即可得到水密化的网格。

6.2 定性对比

图 3列出三个物体在不同方法简化下，同样简化一半面片的结果：

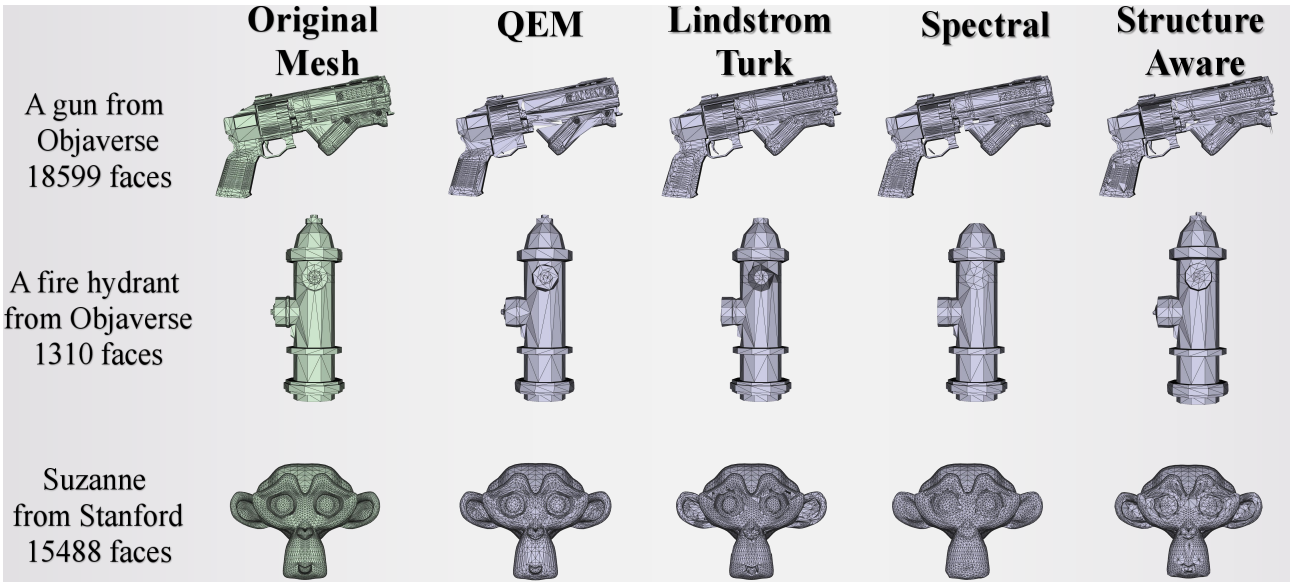

Figure 3: 定性对比的结果

我们可以看出 Lindstrom Turk 以及 Structure Aware 的方法在物体的形状保持上取得了较好的结果，而 Spectral 方法由于更多关注整体特征，因此一些细节在简化过程中丢失。

6.3 定量实验

对于网格的重建效果，我们这里采取了三个指标：CD（Chamfer Distance），EMD（Earth Mover's Distance）以及 SSIM（Structural Similarity Index）。

6.3.1 CD

Chamfer Distance 是衡量两个点云之间相似度的几何距离度量，即为找出一片点云中距离另一片点云中最近点距离平方的加和。这一指标广泛应用于网格重建，点云匹配等工作中。具体计算方法为：给定两个点集 $P = \{p_1, p_2, \ldots, p_m\} \subset \mathbb{R}^d$ 和 $Q = \{q_1, q_2, \ldots, q_n\} \subset \mathbb{R}^d$，具体数值为：

$$\mathrm{CD}(P, Q) = \frac{1}{|P|} \sum_{p \in P} \min_{q \in Q} \|p - q\|_2^2 + \frac{1}{|Q|} \sum_{q \in Q} \min_{p \in P} \|q - p\|_2^2 \tag{1}$$

在我们的实验过程中，我们分别对重建前，重建后的网格表面随机采样 2048 个点，并计算两个点云之间的 Chamfer Distance。

6.3.2 EMD

Earth Mover's Distance 是一种用于衡量两个点集之间“最小搬运代价”的距离度量。可以近似理解为：将一个点云视为“土堆”，另一个点云视为“坑洞”，EMD 衡量的是将一堆土填

满所有坑所需的最小代价。具体方法为：给定两个大小相等的点集 $P = \{p_1, p_2, \ldots, p_N\} \subset \mathbb{R}^d$ 和 $Q = \{q_1, q_2, \ldots, q_N\} \subset \mathbb{R}^d$，Earth Mover's Distance（EMD）定义为：

$$\text{EMD}(P, Q) = \min_{\phi: P \to Q} \frac{1}{N} \sum_{p \in P} \|p - \phi(p)\|_2 \tag{2}$$

在我们的实验过程中，我们分别对重建前，重建后的网格表面随机采样 256 个点，并计算两个点云之间的 Earth Mover's Distance。

6.3.3 SSIM

SSIM 是一种衡量两幅图像结构相似度的感知度量标准，关注图像的结构信息、人眼感知一致性，广泛用于图像质量评估和图像生成任务中。而在网格重建中，也可以作为视觉上渲染结果的衡量指标。具体的计算方式如下：

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3}$$

其中：

- μ_x, μ_y：图像 x、y 的均值；
- σ_x^2, σ_y^2：图像 x、y 的方差；
- σ_{xy}：协方差；
- $C_1 = (K_1 L)^2, \quad C_2 = (K_2 L)^2$

这里我们使用现成的 SSIM 计算方法，对重建前后无色网格在 8 个固定视角进行渲染，并计算重建前后的 8 个 SSIM 的平均值。

6.3.4 结果展示

表 1展示了定性对比的结果，观察可见，Lindstrom Turk 方法在三个指标上均取得了最佳表现，而这也是往后网格简化中最常用的算法。

Metrics	**QEM**	**Lindstrom-Turk**	**Spectral**	**Structure-Aware**
CD ↓	0.0160	**0.0042**	0.0129	0.0094
EMD ↓	0.0347	**0.0226**	0.0314	0.0290
SSIM ↑	0.9477	**0.9674**	0.9319	0.9579

Table 1: 各种方法的定量对比

7 小组分工

- 王麒杰：QEM 算法实现
- 胡昌宇：对比算法调研与实现
- 郭晟宇：前端可视化实现 + 数据集测试实验

References

[1] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[2] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects, 2022.

[3] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. SIGGRAPH '97, page 209–216, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[4] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *CoRR*, abs/2006.12057, 2020.

[5] Thibault Lescoat, Hsueh-Ti Derek Liu, Jean-Marc Thiery, Alec Jacobson, Tamy Boubekeur, and Maks Ovsjanikov. Spectral mesh simplification. *Computer Graphics Forum (Proc. of EUROGRAPHICS 2020)*, 39(2):315–324, 2020.

[6] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *Proceedings of the Conference on Visualization '98*, VIS '98, page 279 – 286, Washington, DC, USA, 1998. IEEE Computer Society Press.

[7] Hsueh-Ti Derek Liu and Alec Jacobson. Cubic stylization. *ACM Transactions on Graphics*, 2019.

[8] Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *Proceedings of the IEEE international conference on computer vision*, pages 2353–2361, 2017.

[9] D. Salinas, F. Lafarge, and P. Alliez. Structure-aware mesh decimation. *Comput. Graph. Forum*, 34(6):211–227, September 2015.