1 几何表示

1.1 几何形状的表示方法

几何形状 (geometric shape) 是指空间中的一组特定点的集合. 对于二维空间, 常见的几何图形包括线段, 多边形, 圆等等; 对于三维空间, 常见的几何图形包括线, 面, 体等.

1.1.1 连续函数表示法

和平面图形一样, 空间图形也可以用连续函数表示. 例如, 空间中的直线可以表示为

$$\{(x,y,z) : Ax + By + Cz + D = 0, x, y, z \in \mathbb{R}\}$$

空间中的球面可以表示为

$$\left\{(x,y,z) : x^2 + y^2 + z^2 = R^2, x, y, z \in \mathbb{R}\right\}$$

定义 1.1 连续函数表示法 一般地, 对于空间中的几何形状 M, 可以用一个特定的连续函数 $S_M(x,y,z)$ 刻画其表面 ∂M:

$$\{(x,y,z) : S_M(x,y,z) = 0, x, y, z \in \mathbb{R}\}$$

连续函数可以精确地刻画几何形状, 也方便研究其性质. 然而, 对于复杂的图形 (尤其是复杂的曲线或曲面), 难以找到合适的函数来表示; 并且这一表达形式是隐式的, 不能直接将几何形状呈现出来. 这就需要离散化的办法.

1.1.2 点云

通过类似雷达的工作方式对几何形状 M 进行扫描, 可以获知 ∂M 上一系列离散的点.

定义 1.2 点云 **点云 (Point Cloud)** 是一组三维空间中有限个点构成的集合:

$$\{(x_i, y_i, z_i) : i = 1, \cdots, N\}$$

这集合描述的几何形状 M 满足: 对任意点云中的点 (x_i, y_i, z_i), 都有 $(x_i, y_i, z_i) \in \partial M$, 即

$$S_M(x_i, y_i, z_i) = 0$$

点云作为原本几何形状的采样结果, 保留了原形状的一部分几何信息，, 使得我们可以在点云上进行一些表面性质的计算, 例如计算法向和曲率.

然而采样总是伴随着信息的损失, 点云也不例外. 点云的采样密度决定了它对几何细节的表示能力. 更重要的是, 由于点云本身的非结构化和无序性, 几何形状的拓扑关系往往是最容易在点云表示中变得模糊不清的. 这为基于点云的几何形状计算和处理带来了困难.

1.1.3 网格模型

为了解决点云对于拓扑形状表示的不足, 我们考虑对曲面表面进行线性近似, 即用一系列小的多边形片段拼接成曲面. 为此, 把点云中的点按照 M 的形状进行连接, 可以得到一个由多边形构成的网格.

定义 1.3 网格模型 **网格模型 (Mesh Model)** 是由一组顶点, 边和面构成的三元组:

$$\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$$

其中 $\mathcal{V} = \{\boldsymbol{v}_i = (x_i, y_i, z_i) : i = 1, \cdots, N\}$ 是顶点集合, 每个顶点对应点云中的一个点; $\mathcal{E} = \{\boldsymbol{e}_{ij} = (v_i, v_j) : i, j = 1, \cdots, N\}$ 是边集合, 每条边连接两个顶点; $\mathcal{F} = \{\boldsymbol{f}_k\}$ 是面集合, 每个面 $\boldsymbol{f}_k$ 由数个顶点组成.
$\mathcal{M}$ 描述的几何形状满足: 对于任意面 $\boldsymbol{f}_k$, 其上所有点都在 ∂M 上, 即

$$\forall (x_k, y_k, z_k) \in \boldsymbol{f}_k, \quad S_M(x_k, y_k, z_k) = 0$$

一般而言, 我们要求网格是**流形**的, 即每条边最多被两个面共享. 有时, 我们还要求网格是**水密**的, 即一个由封闭曲面组成的, 没有孔洞的网格.

1.2 网格表示

网格模型是计算机图形学中最常用的几何表示方法, 我们已经在上一节介绍过其定义. 在计算机中存储时, 通常使用顶点列表, 边列表和面列表作为储存多边形网格的数据结构; 有时也会设计额外的数据结构方便邻边查找等操作.

1.2.1 三角网格表示法

我们从最简单的三角网格开始, 即所有面都是三角形的网格. 最简单的表示方法就是记录每个三角形的三个顶点, 即

$$\triangle_i = (\boldsymbol{v}_{i0}, \boldsymbol{v}_{i1}, \boldsymbol{v}_{i2}), \quad \text{where } \boldsymbol{v}_{ij} = (x_{ij}, y_{ij}, z_{ij})$$

定义 1.4 三角形乱序集合 **三角形乱序集合 (Triangle Soup)** 是由一组三角形构成的集合, 每个三角形包括其顶点信息.

显然, 每个顶点几乎都会被多个三角形共用, 因此上面的表示方法在空间上由很大冗余. 并且由于存储的乱序性, 我们也不易对模型进行拓扑关系的考察.

为了减少空间开销, 我们可以先单独存储顶点, 然后只存储每个三角形顶点在顶点列表中的索引.

定义 1.5 索引三角形网格 **索引三角形网格 (Indexed Triangle Set)** 由顶点列表 $\mathcal{V}$ 和三角形列表 $\mathcal{F}$ 构成. 其中 $\mathcal{V} = \{\boldsymbol{v}_i = (x_i, y_i, z_i) : i = 1, \cdots, N\}$ 是顶点列表; $\mathcal{F} = \{\triangle_k\}$ 是三角形列表, 每个三角形 $\triangle_k$ 由三个顶点索引组成.

特别地, 为了方便处理, 我们在存储顶点索引时可以按照逆时针方向存储. 这可以保证每个三角形的法向方向一致, 从而方便后续的渲染等操作.

1.2.2 半边数据结构

在网格中, 我们经常会面对顶点邻接关系的查询, 也需要有序地遍历顶点和面. 在一般的索引三角形网格中, 我们只能通过遍历所有三角形来找到某个顶点的邻接顶点, 这显然效率很低. **半边数据结构 (Half-Edge Data Structure)** 就是一种更高效的网格表示方法.

由于我们主要考虑流形, 因此可以把每条边拆成两个方向相反的**半边 (half-edge)**, 每个半边只属于一个面, 而每个面可以由首尾相接的数个半边表示. 对于每个半边, 我们记录其起点, 终点, 所属面, 上一个和下一个半边, 以及与其配对的另一个半边. 这样, 我们就可以通过半边快速地找到顶点的邻接顶点和邻接面.

定义 1.6 半边数据结构 **半边数据结构 (Half-Edge Data Structure)** 相比一般的网格数据结构增加了对每个边的半边表示. 每个半边 `e` 主要包含以下信息:

- `e->From()`: 半边的起点;
- `e->Twin()`: 与该半边配对的另一个半边;
- `e->Face()`: 该半边所属的面;
- `e->Next()`: 该半边在所属面中的下一个半边;
- `e->Prev`: 该半边在所属面中的上一个半边.
- `e->To()`: 半边的终点, 等于 `e->Twin()->From()`.

实现半边数据结构主要通过**双向边链表 (DCEL, Doubly Connected Edge List)** 来完成. DECL 中不仅新增了半边的信息, 其面和顶点也储存了与相关半边的信息, 从而构成了一张结构完善的图.

使用半边数据结构能完成涉及网格拓扑结构的各种操作, 例如. 下面给出两个简单的例子.

遍历面的顶点/边 给定面 `f`, 可以通过 `f->Edge()`[1]得到该面上的一个半边 `e`. 然后不断访问 `e->Next()`, 直到回到起点, 就可以遍历该面上的所有顶点和边.

```
1 DCEL::HalfEdge const * e = f->Edge();
2 DCEL::HalfEdge const * e_start = f->Edge();
3 do {
4     DCEL::VertexProxy const * v = e->From(); // 访问顶点
5     e = e->Next(); // 访问下一条边
```

[1]在 Lab 中可以直接通过 `f->Edge(i)` 访问顶点 `i` 的对边, 但这里采用更加本质的办法, 即通过类似环状链表的形式遍历.

```
} while (e != e_start);
```

围绕顶点进行遍历 前面我们给出了围绕面构造迭代器的办法. 另一个常用的迭代器是顶点迭代器, 即围绕某个顶点访问其所有邻接顶点[2]. 给定顶点 `v`, 可以通过 `v->Edge()` 得到一条以 `v` 为起点的半边 `e`. 然后不断访问 `e->Twin()->Next()`, 直到回到起点, 就可以遍历该顶点的所有邻接顶点.

```
DCEL::HalfEdge const * e = v->Edge();
DCEL::HalfEdge const * e_start = v->Edge();
do {
    DCEL::VertexProxy const * u = e->To(); // 访问邻接顶点
    e = e->Twin()->Next(); // 访问下一条邻接顶点的边
} while (e != e_start);
```

总之, 半边数据结构通过增加对边的表示, 使得网格的拓扑关系更加清晰, 从而方便了各种基于网格的计算和处理.

1.3 网格细分

通过增加组成几何表面的网格面片数量, 减小每个面片的面积, 可以使几何表面看起来更加光滑.

定义 1.7 网格细分 **网格细分 (Mesh Subdivision)**, 又称作网格的**上采样**, 是指通过反复细分初始的多边形网格, 不断得到更精细的网格的过程.

1.3.1 Catmull-Clark 细分

Catmull-Clark 细分是最常用的几何表面细分方法之一, 主要应用于四边形网格的细分上. Catmull-Clark 细分的具体步骤如下:

1. **增设面点**: 对多面体的每个面片计算一个面点, 该面点是该面片所有顶点的平均值:

$$\boldsymbol{v}_{\text{face}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{v}_n$$

2. **增设边点**: 对多面体的每条边计算一个边点, 该边点是该边两个端点 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$, 以及相邻两个面片的面点 $\boldsymbol{v}_{\text{face},1}$ 和 $\boldsymbol{v}_{\text{face},2}$ 的平均值:

$$\boldsymbol{v}_{\text{edge}} = \frac{1}{4}\left(\boldsymbol{v}_1 + \boldsymbol{v}_2 + \boldsymbol{v}_{\text{face},1} + \boldsymbol{v}_{\text{face},2}\right)$$

3. **更新顶点**: 对多面体原有的每个顶点 $\boldsymbol{v}$, 使用下面的加权平均算法更新其位置:

$$\boldsymbol{v}_{\text{vertex}} = \frac{\boldsymbol{F} + 2\boldsymbol{R} + (N-3)\boldsymbol{v}}{N}$$

[2]在 Lab 中, 可以通过 `v->Ring()` 直接访问所有相邻的面中与 `v` 不相交的, 相对的半边. 对于非边界上的 `v`, 这些半边组成了一个环.

其中 $\boldsymbol{F}$ 是所有以 $\boldsymbol{v}$ 为顶点的面片的面点的均值, $\boldsymbol{R}$ 是所有以 $\boldsymbol{v}$ 为端点的边的中点 (注意不是 **2.** 中的边点) 的均值, N 是与该顶点相邻的顶点数.

4. **重新连接**: 将每个面点 $\boldsymbol{v}_{\text{face}}$ 与该面片所有边对应的的边点 $\boldsymbol{v}_{\text{edge}}$ 相连; 将每个新顶点 $\boldsymbol{v}_{\text{vertex}}$ 与原有顶点所有相邻边的边点 $\boldsymbol{v}_{\text{edge}}$ 相连. 于是就形成新的细分过后的面片.

1.3.2 Loop 细分

Loop 细分是另一种常用的几何表面细分方法, 主要应用于三角形网格的细分上. Loop 细分的具体步骤如下:

1. **增设新顶点**: 对于每一条边, 如果这条边被两个三角形面包含, 则根据这条边的两个端点 $\boldsymbol{v}_0, \boldsymbol{v}_2$ 和这两个三角形除这条边外各自的顶点 $\boldsymbol{v}_1, \boldsymbol{v}_3$ 加权平均得到新顶点 $\boldsymbol{v}^*$:

$$\boldsymbol{v}^* = \frac{3}{8}(\boldsymbol{v}_0 + \boldsymbol{v}_2) + \frac{1}{8}(\boldsymbol{v}_1 + \boldsymbol{v}_3)$$

   如果这条边只被一个三角形面包含, 则取边的中点得到新顶点 $\boldsymbol{v}^*$.

2. **更新原有顶点**: 对于每个原有的顶点 $\boldsymbol{v}$, 按照下面的公式更新其位置至 $\boldsymbol{v}'$:

$$\boldsymbol{v}' = (1 - nu)\boldsymbol{v} + \sum_{i=1}^{n} u\boldsymbol{v}_i$$

   其中 n 为 $\boldsymbol{v}$ 邻接顶点的数目,$\boldsymbol{v}_i$ 是与 $\boldsymbol{v}$ 相邻的顶点. 当 $n = 3$ 时,$u = \dfrac{3}{16}$; 当 $n > 3$ 时,$u = \dfrac{3}{8n}$.

3. **重新连接**: 将每个原有三角形的三个边的构造出的新点与原有顶点更新后的位置相连, 形成四个新的三角形.

1.4 网格参数化

1.4.1 网格参数化概述

网格参数化起源于纹理映射的需要. 在为三维图形着色时 (类似地, 将三维图形展平至二维, 例如绘制世界地图等情形时), 我们需要建立三维图形表面与二维纹理图像之间的映射关系, 即建立三维图形上的点 $\boldsymbol{v}_i(x_i, y_i, z_i)$ 到二维平面上的点 $\boldsymbol{u}_i(u_i, v_i)$ 的映射 (这与 UV 坐标的思想类似).

定义 1.8 网格参数化 **网格参数化 (Mesh Parameterization)** 是指将三维空间中的网格映射到二维平面上的过程. 具体地, 对于三维空间中的网格 $\mathcal{M} =$, 我们希望找到一个映射 $f : \mathbb{R}^3 \to \mathbb{R}^2$, 使得每个顶点 $\boldsymbol{v}_i \in \mathcal{V}$ 都对应一个二维平面上的点 $\boldsymbol{u}_i = f(\boldsymbol{v}_i) \in \mathbb{R}^2$.

按照参数化中保持的几何量, 可以将网格参数化分为保长度 (Isometric) 的参数化, 保角 (Conformal) 的参数化和保面积 (Areal) 的参数化等. 保长度的参数化等价于既保角度又保面积的参数化. 理想的参数化可以保持形状不发生变化, 但只有可展曲面 (例如圆柱面) 可以进行保形参数化, 而大部分曲面在参数化时都会发生一定形变.

一般而言, 我们参数化的对象是具有边界的开网格. 对于封闭的图形, 通常先指定一条边将其裁切为开网格, 然后进行参数化.

对开网格的参数化主要分为**固定边界映射**和**自由边界映射**两类. 固定边界映射是指将边界顶点映射到二维平面的一个预设形状 (例如圆形或正方形) 上, 然后计算内部顶点映射到的坐标. 自由边界映射则不对边界顶点进行预设, 而是同时优化确定边界和内部顶点的二维坐标.

1.4.2 基于弹簧模型的网格参数化/重心映射

在开网格参数化中, 最基础的模型为**弹簧模型**. 我们将网格顶点作为节点, 网格的边作为连接节点的弹簧. 参数化后的系统状态, 可以由各个弹簧的弹性势能之和衡量, 最终的平衡状态 (系统总弹性势能最小的状态) 即为参数化的结果.

考虑网格 $\mathcal{M}$ 的顶点 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$, 各顶点参数化后的二维坐标为 $\boldsymbol{t}_1, \cdots, \boldsymbol{t}_n$. 系统的总能量 (求和时每条边的能量都被计算两遍, 因此前面还需乘以 1/2) 可以描述为

$$E = \frac{1}{2}\sum_{i=1}^{n}\sum_{j\in\Omega(i)}\frac{1}{2}D_{ij}\left|\left|\boldsymbol{t}_i - \boldsymbol{t}_j\right|\right|^2$$

其中 $\Omega(i)$ 表示与顶点 i 相邻的顶点集合, D_{ij} 是连接顶点 i 和 j 的弹簧的弹性系数. 系统能量最小时, 对各 $\boldsymbol{t}_i$ 的偏导均为 $\mathbf{0}$, 即

$$\frac{\partial E}{\partial \boldsymbol{t}_i} = \sum_{j\in\Omega(i)} D_{ij}\left(\boldsymbol{t}_i - \boldsymbol{t}_j\right) = \mathbf{0}$$

记组合系数

$$\lambda_{ij} = \frac{D_{ij}}{\sum\limits_{k\in\Omega(i)} D_{ik}}$$

于是上式可以整理为

$$\boldsymbol{t}_i - \sum_{j\in\Omega(i)} \lambda_{ij}\boldsymbol{t}_j = \mathbf{0}$$

可以看出,$\boldsymbol{t}_i$ 由其周围的点按照一定系数加权得到, 因此这一方法又称作**重心映射**. 这是一个有 $2n$ 个变量和 $2n$ 个方程的齐次线性方程组, 其平凡解为零解. 为了得到非零解, 一种简单的办法是将开网格边界上的点根据它们之间的距离映射到指定的凸图形的边界 (常见的有正方形, 圆形) 上, 然后将内部的点作为未知量按照上面的方程组求解.

另外, 可以设置合适的系数 λ_{ij} 引导参数化. 系数的设置需满足

1. **凸组合性**: 对凸多边形的加权平均仍然在凸多边形内部, 即

$$\lambda_{ij} \geqslant 0, \quad \sum_{j\in\Omega(i)} \lambda_{ij} = 1$$

2. **线性重构性**: 如果顶点 $\boldsymbol{v}_i$ 和其邻接顶点 $\boldsymbol{v}_j$ 在三维空间中共线, 则参数化后的点 $\boldsymbol{t}_i$ 和 $\boldsymbol{t}_j$ 也应当共线; 亦即平面网格映射后保持不变. 即

$$\sum_{j\in\Omega(i)} \lambda_{ij}\left(\boldsymbol{v}_i - \boldsymbol{v}_j\right) = \mathbf{0}$$

基于此和一些几何上的考虑, 常见的组合系数有下面几种:

1. **平均系数**: 每个邻接顶点的权重相等, 即
$$\lambda_{ij} = \frac{1}{|\Omega(i)|}$$
2. **均值坐标系数 (Floater 权重)**: 基于顶点 $\boldsymbol{v}_i$ 和其邻接顶点 $\boldsymbol{v}_j$ 与 $\boldsymbol{v}_i$ 的连线与其邻接边的夹角 α_{j1} 和 α_{j2}, 定义权重为
$$\lambda_{ij} = \frac{\tan(\alpha_{j1}/2) + \tan(\alpha_{j2}/2)}{||\boldsymbol{v}_i - \boldsymbol{v}_j||}$$
3. **调和坐标系数 (余切 Laplacian 权重)**: 基于顶点 $\boldsymbol{v}_i$ 和其邻接顶点 $\boldsymbol{v}_j$ 与 $\boldsymbol{v}_i$ 的连线相对的夹角 β_{j1} 和 β_{j2}, 定义权重为
$$\lambda_{ij} = \frac{\cot\beta_{j1} + \cot\beta_{j2}}{2}$$
调和坐标系数更接近保角映射, 但也可能出现权重为负值的情况, 从而导致映射出现折叠.