1 几何重建

1.1 几何重建概述

现实世界中的三维物体和场景无法直接以数学模型或几何形式被计算机存储或处理. 大多数时候, 我们只能通过图像, 视频或点云数据间接地获取三维信息, 这些数据本身离散而稀疏, 带有噪声, 也没有拓扑结构. 为了用计算机处理形状, 空间结构和几何关系, 必须将这些数据重建为连续的几何模型.

定义 1.1 几何重建 **几何重建 (Geometry Reconstruction)** 从离散的三维数据 (如点云, 深度图像或多视图图像) 中, 重建出物体或场景的连续几何形状与结构的过程.

1.2 几何变换基础

定理 1.2 平移 平移对应的数学操作为向量加法. 设物体上任意一点 P 的坐标为 $\boldsymbol{p}$, 平移向量为 $\boldsymbol{t}$, 则平移后的点 P' 的坐标 $\boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{t}$. 例如, 对于三维空间而言有

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \end{bmatrix}$$

定理 1.3 旋转 旋转对应的数学操作为矩阵乘法. 设物体上任意一点 P 的坐标为 $\boldsymbol{p}$, 旋转矩阵为 $\boldsymbol{R}$, 则旋转后的点 P' 的坐标 $\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p}$. 在二维平面中, 绕原点逆时针旋转 θ 角度的旋转矩阵为

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

在三维空间中, 绕 x, y, z 轴分别旋转 α, β, γ 角度的旋转矩阵分别为

$$\boldsymbol{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \quad \boldsymbol{R}_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, \quad \boldsymbol{R}_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

定理 1.4 绕任意轴的旋转 在三维空间中, 绕单位向量 $\boldsymbol{a} = (a_x, a_y, a_z)$ 定义的轴逆时针旋转 θ 角度的旋转矩阵为

1.3 点云的配准/注册

在几何重建中, 我们往往会从不同角度和位置对同一场景或物体分别采集点云数据. 将不同视角下的点云对齐到同一坐标系下, 才能得到完整的重建数据.

定义 1.5 点云配准/注册 **点云配准/注册 (Point Cloud Registration)** 是将来自不同视角或传感器的多个点云数据集对齐到同一坐标系下的过程.

现在我们来介绍一种经典的点云配准算法, 即 ICP(Iterative Closest Point) 算法.

1.3.1 点云配准的 ICP 算法

我们在不同角度测得的点云总是保长度的, 这意味着描述同一物体的点云之间总是可以通过平移和旋转变换对齐.

给定待变换点云 S 和目标点云 T, ICP 算法的目标是找到一组最佳的旋转和平移变换 $\boldsymbol{R}, \boldsymbol{t}$ 使得 S 经变换后最大程度上地与 T 对其. ICP 算法的具体流程如下:

1. 通过主成分分析 (PCA, Principal Component Analysis) 初始化一组变换 $\boldsymbol{R}, \boldsymbol{t}$ 作为估计变换的初值.

2. 将当前求得的变换 $\boldsymbol{R}, \boldsymbol{t}$ 应用于 S, 对于 S 中的各点 $\boldsymbol{p}_i$ 变换后得到 $\boldsymbol{p}'_i = \boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t}$. 然后找到 T 中与 $\boldsymbol{p}'_i$ 中最接近的点 $\boldsymbol{q}_i$, 如此构成 S' 与 T 的一一对应. 同时, 舍去距离过远的点对.

3. 在余下的 N 组点对 $\{(\boldsymbol{p}'_i, \boldsymbol{q}_i) : i = 1, \cdots, N\}$ 中, 构造累计误差函数

$$E = \sum_{i=1}^{N} ||\boldsymbol{p}'_i - \boldsymbol{q}_i||^2 = \sum_{i=1}^{N} ||\boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t} - \boldsymbol{q}_i||^2$$

注意这里的 $\boldsymbol{R}, \boldsymbol{t}$ 是需要求解的变量, 与 **2.** 中前一步迭代求得的 $\boldsymbol{R}, \boldsymbol{t}$ 是不同的. 前一步求得的 $\boldsymbol{R}, \boldsymbol{t}$ 只用于对应关系的确定, 不直接参与后续的最小化计算.

然后, 通过奇异值分解 (SVD, Singular value decomposition) 求解最小化问题:

$$(\boldsymbol{R}, \boldsymbol{t}) = \arg\min_{\boldsymbol{R}, \boldsymbol{t}} E$$

4. 重复步骤 2 和 3, 直到累计误差 E 小于预设阈值或达到最大迭代次数.

现在我们具体地介绍上述算法中重要的两步, 即 PCA 和 SVD.

1.3.2 主成分分析

定义 1.6 主成分分析 **主成分分析 (Principal Component Analysis, PCA)** 是一种统计方法, 用于通过线性变换将数据从高维空间映射到低维空间, 以保留数据的主要特征和结构.

一般而言, 点云的分布在空间中是各向异性的, 即在某些方向上点云的变化更显著. 通过 PCA 可以找到点云数据的主要方向 (即主轴), 从而为 ICP 算法提供一个合理的初始变换估计. 现在给出 PCA 的具体步骤.

首先, 考虑给定的一组点[1] $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N$ 以及中心坐标

$$\bar{\boldsymbol{p}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{p}_i$$

首先, 为了防止平移对 PCA 结果的影响, 我们需要将点云数据进行中心化处理, 即将每个点减去中心坐标:

$$\boldsymbol{p}_i' = \boldsymbol{p}_i - \bar{\boldsymbol{p}}, \quad i = 1, \cdots, N$$

构造矩阵

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}_1' \\ \boldsymbol{p}_2' \\ \vdots \\ \boldsymbol{p}_N' \end{bmatrix}$$

然后计算 $\boldsymbol{P}$ 的协方差矩阵

$$\boldsymbol{\Sigma}_{\boldsymbol{P}} = \boldsymbol{P}\boldsymbol{P}^{\mathrm{t}}$$

二维情况下的协方差矩阵的两个特征向量 $\boldsymbol{v}_1, \boldsymbol{v}_2$ 表示数据的两条轴线. 对应于更大的特征值的特征向量 $\boldsymbol{v}_1$ 表示数据变化更显著的方向, 即主成分方向, 而 $\boldsymbol{v}_2$ 则表示数据变化最小的方向[2]. 对于三维情况也是同理.

对协方差矩阵 $\boldsymbol{\Sigma}_{\boldsymbol{P}}$ 进行特征值分解, 可得:

$$\boldsymbol{\Sigma}_{\boldsymbol{P}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{t}}$$

其中矩阵 $\boldsymbol{V}$ 的列向量即为 $\boldsymbol{\Sigma}_{\boldsymbol{P}}$ 的特征向量, 它们也就确定了点云的主轴方向.

现在, 对给定的两个点云 S 和 T, 我们分别对它们进行 PCA, 得到各自的中心 $\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}$ 以及主轴 $\boldsymbol{v}_{s1}, \boldsymbol{v}_{s2}, \boldsymbol{v}_{s3}$ 和 $\boldsymbol{v}_{t1}, \boldsymbol{v}_{t2}, \boldsymbol{v}_{t3}$. 既然变换仅由平移和旋转构成, 我们可以认为 S 和 T 的主轴应当近似地重合. 于是, 先通过平移使得两个点云的中心重合, 即

$$\boldsymbol{t} = \bar{\boldsymbol{q}} - \bar{\boldsymbol{p}}$$

然后, 通过旋转使得两个点云的主轴方向对齐. 设旋转矩阵为 $\boldsymbol{R}$, 则有

$$\boldsymbol{R}\boldsymbol{v}_{si} = \boldsymbol{v}_{ti}, \quad i = 1, 2, 3$$

这样求得的 $\boldsymbol{R}, \boldsymbol{t}$ 即可作为 ICP 算法的初始变换估计.

1.3.3 奇异值分解

对于前述的最小化问题, 事实上是具有唯一解的. 我们可以用奇异值分解来求解.

注意到 E 中含有两个变量 $\boldsymbol{R}, \boldsymbol{t}$. 我们先求解使得 E 最小的 $\boldsymbol{t}$, 然后再求解 $\boldsymbol{R}$. 这是为了对齐质心后才能进

[1] 每个点都是一个三维向量, 表示该点在三维空间中的位置.

[2] 例如, 如果数据点沿一条直线附近随机分布, 那么 $\boldsymbol{v}_1$ 就是该直线的法向量,$\boldsymbol{v}_2$ 是该直线的方向向量

行旋转矩阵的求解.

设 S 和 T 的中心分别为 $\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}$. 令

$$\tilde{\boldsymbol{p}}_i = \boldsymbol{p}_i - \bar{\boldsymbol{p}}, \quad \tilde{\boldsymbol{q}}_i = \boldsymbol{q}_i - \bar{\boldsymbol{q}}$$

于是

$$E = \sum_{i=1}^{N} ||\boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t} - \boldsymbol{q}_i||^2 = \sum_{i=1}^{N} ||\boldsymbol{R}\tilde{\boldsymbol{p}}_i - \tilde{\boldsymbol{q}}_i + (\boldsymbol{R}\bar{\boldsymbol{p}} + \boldsymbol{t} - \bar{\boldsymbol{q}})||^2$$

关于 $\boldsymbol{R}\bar{\boldsymbol{p}} + \boldsymbol{t} - \bar{\boldsymbol{q}}$ 展开后对 $\boldsymbol{t}$ 求导并令其为 0, 可得 (实际上直接观察也容易得到)

$$\boldsymbol{R}\bar{\boldsymbol{p}} + \boldsymbol{t} - \bar{\boldsymbol{q}} = \boldsymbol{0}$$

于是

$$\boldsymbol{t} = \bar{\boldsymbol{q}} - \boldsymbol{R}\bar{\boldsymbol{p}}$$

注意这里的 $\boldsymbol{t}$ 是关于 $\boldsymbol{R}$ 的函数, 因此求解 $\boldsymbol{R}$ 完毕后需要重新进行带入. 现在把 $\boldsymbol{t}$ 代入 E, 可得

$$E = \sum_{i=1}^{N} ||\boldsymbol{R}\tilde{\boldsymbol{p}}_i - \tilde{\boldsymbol{q}}_i||^2$$

上面的式子中已经没有 $\boldsymbol{t}$, 即已经完成对齐质心的工作. 我们只需要最小化 E 关于 $\boldsymbol{R}$ 的部分. 将平方展开, 并且注意到 $||\boldsymbol{R}\tilde{\boldsymbol{p}}_i|| = ||\tilde{\boldsymbol{p}}_i||$(旋转显然不改变向量的模长), 于是可得

$$E = \sum_{i=1}^{N} \left(||\tilde{\boldsymbol{p}}_i||^2 + ||\tilde{\boldsymbol{q}}_i||^2 \right) - 2\sum_{i=1}^{N} \tilde{\boldsymbol{q}}_i^{\mathrm{t}} \boldsymbol{R}\tilde{\boldsymbol{p}}_i$$

注意到上式中只有最后一项与 $\boldsymbol{R}$ 有关, 因此等价于最大化

$$\boldsymbol{R}^* = \arg\max_{\boldsymbol{R}} \sum_{i=1}^{N} \tilde{\boldsymbol{q}}_i^{\mathrm{t}} \boldsymbol{R}\tilde{\boldsymbol{p}}_i$$

自然地, 我们会想到标量与矩阵的迹之间的关联. 由于每个求和项都是向量的内积, 可以视作 1×1 的矩阵, 于是自然可以将它写做迹的形式:

$$\tilde{\boldsymbol{q}}_i^{\mathrm{t}} \boldsymbol{R}\tilde{\boldsymbol{p}}_i = \mathrm{trace}\left(\tilde{\boldsymbol{q}}_i^{\mathrm{t}} \boldsymbol{R}\tilde{\boldsymbol{p}}_i\right) = \mathrm{trace}\left(\tilde{\boldsymbol{p}}_i \tilde{\boldsymbol{q}}_i^{\mathrm{t}} \boldsymbol{R}\right)$$

于是求和后就有

$$\sum_{i=1}^{N} \tilde{\boldsymbol{q}}_i^{\mathrm{t}} \boldsymbol{R}\tilde{\boldsymbol{p}}_i = \mathrm{trace}\left(\sum_{i=1}^{N} \tilde{\boldsymbol{p}}_i \tilde{\boldsymbol{q}}_i^{\mathrm{t}} \boldsymbol{R}\right) = \mathrm{trace}\left(\boldsymbol{R}\sum_{i=1}^{N} \tilde{\boldsymbol{p}}_i \tilde{\boldsymbol{q}}_i^{\mathrm{t}}\right)$$

为了写成矩阵形式, 定义

$$\boldsymbol{M} = \sum_{i=1}^{N} \tilde{\boldsymbol{p}}_i \tilde{\boldsymbol{q}}_i^{\mathrm{t}}$$

这是一个 3×3 的矩阵, 对其进行奇异值分解, 可得

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{t}}$$

其中 $\boldsymbol{U}, \boldsymbol{V}$ 是正交矩阵, $\boldsymbol{\Sigma}$ 是对角矩阵, 其对角线元素为非负的奇异值. 于是

$$\operatorname{trace}\left(\boldsymbol{R}\sum_{i=1}^{N}\tilde{\boldsymbol{p}}_i\tilde{\boldsymbol{q}}_i^{\mathrm{t}}\right)=\operatorname{trace}\left(\boldsymbol{R}\boldsymbol{M}\right)=\operatorname{trace}\left(\boldsymbol{R}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{t}}\right)=\operatorname{trace}\left(\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{t}}\boldsymbol{R}\boldsymbol{U}\right)$$

由于 $\boldsymbol{R}$ 是旋转矩阵, 因此它自然也是正交矩阵. 于是设 $\boldsymbol{X}=\boldsymbol{V}^{\mathrm{t}}\boldsymbol{R}\boldsymbol{U}$, 则 $\boldsymbol{X}$ 也是正交矩阵. 于是

$$\operatorname{trace}\left(\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{t}}\boldsymbol{R}\boldsymbol{U}\right)=\operatorname{trace}\left(\boldsymbol{\Sigma}\boldsymbol{X}\right)=\sigma_1x_{11}+\sigma_2x_{22}+\sigma_3x_{33}$$

其中 σ_i 为 $\boldsymbol{M}$ 的奇异值 (即 $\boldsymbol{\Sigma}$ 的对角线元素), x_{ii} 为 $\boldsymbol{X}$ 的对角线元素. 由于 $\boldsymbol{X}$ 是正交矩阵, 因此其元素的绝对值均不大于 1, 即 $|x_{ii}|\leqslant 1$. 为了最大化上式, 显然需要 $x_{ii}=1$. 这就要求 $\boldsymbol{X}=\boldsymbol{I}$, 即

$$\boldsymbol{V}^{\mathrm{t}}\boldsymbol{R}\boldsymbol{U}=\boldsymbol{I}\Rightarrow\boldsymbol{R}=\boldsymbol{V}\boldsymbol{U}^{\mathrm{t}}$$

最后, 还需确保 $\boldsymbol{R}$ 是合法的旋转矩阵, 即要求 $\det\boldsymbol{R}=1$. 如果 $\det\boldsymbol{R}=-1$, 这样的 $\boldsymbol{R}$ 表示了一个反射变换. 这里直接给出修正后的结果 (这涉及到一些线性代数知识):

$$\boldsymbol{R}^*=\boldsymbol{V}\begin{bmatrix}1&0&0\\0&1&0\\0&0&\det\boldsymbol{V}\boldsymbol{U}^{\mathrm{t}}\end{bmatrix}=\boldsymbol{U}^{\mathrm{t}}$$

当然, 在课程中并没有讲到这种情况, 因此有

$$\boldsymbol{R}^*=\boldsymbol{V}\boldsymbol{U}^{\mathrm{t}}$$

回代求出 $\boldsymbol{t}^*$, 有

$$\boldsymbol{t}^*=\bar{\boldsymbol{q}}-\boldsymbol{R}^*\bar{\boldsymbol{p}}$$

这样得到的 $\boldsymbol{R}^*, \boldsymbol{t}^*$ 即可作为下一轮迭代用于确定对应关系的变换.

1.4 点云的表面重建

1.4.1 Voronoi 图

对于给定的点云 S, 一种自然地划分空间的方式是将空间中每个点分配给距离其最近的点云中的点.

定义 1.7 Voronoi 图 **Voronoi 图 (Voronoi Diagram)** 是根据点集 P 将空间划分为若干个区域的几何结构, 其中每个区域 Ω_i 对应于 P 中一个给定的顶点 $\boldsymbol{v}_i$, 使得 Ω_i 内的所有点都比 P 中其他顶点 $\boldsymbol{v}_j$ 更接近该顶点 $\boldsymbol{v}_i$.

事实上, 在二维情形中的 Voronoi 图正是每个点与它临近的点的垂直平分线所构成的图形. 尽管如此的定义很简单, 但在实现上却仍比较复杂. 我们可以用另外一种方式间接地实现 Voronoi 图的构造.

1.4.2 Delaunay 三角剖分

定义 1.8 Delaunay 三角剖分 **Delaunay 三角剖分 (Delaunay Triangulation)** 是点集 $\mathcal{P}$ 的剖分, 它将 $\mathcal{P}$ 连接成三角形网格, 使得每个三角形的外接圆内不包含其他点.

Delaunay 三角剖分是 Voronoi 图的对偶图.Voronoi 图的每个顶点对应 Delaunay 三角剖分中的一个三角形的外心, Delaunay 三角剖分中的每个顶点即为 Voronoi 图中的一个区域对应的点.

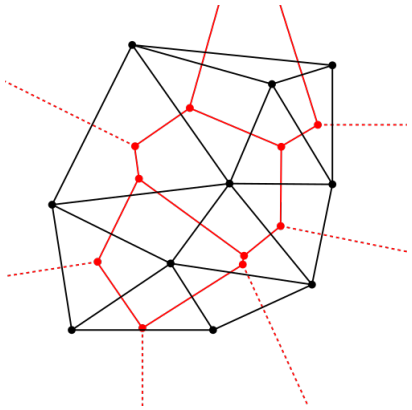

图 1: Delaunay 三角剖分与 Voronoi 图的关系

可以证明 Delaunay 三角剖分是使得所有三角形中最小内角最大化的剖分方式, 因而尽可能避免了细长三角形的出现. 结合三角形外接圆的形式, 我们可以得出: 在 Delaunay 三角剖分中, 任一被两个三角形共用的边所对的两个角之和小于 180°(否则这四点将落在同一圆内, 这是与定义矛盾的.). 然而, 如果将这一条边所在的四边形替换为由另一条对角线所构成的两个三角形, 则这两个新三角形就符合剖分性质了. 据此, 我们可以定义 Delaunay 三角剖分中最基本的操作——边翻转 (Edge Flip), 如下图所示:

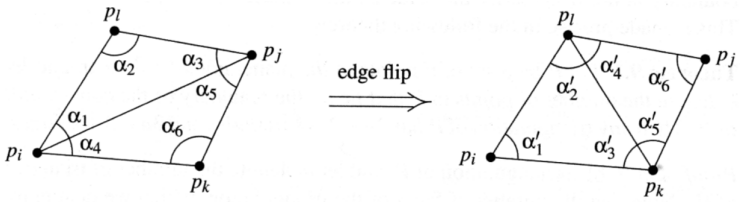

图 2: 边翻转操作

于是, 这样的思路产生了一个非常简单的算法: 任一构造一个三角剖分, 检查所有被共用的边, 如果不满足上述条件就进行边翻转操作, 直到所有边都满足条件为止.

遗憾的是这种算法效率并不高. 我们可以采取增量式算法, 每次向已有的三角剖分中加入一个点, 并检查受影响的三角形和进行必要的边翻转操作. 效率最高的算法是分治算法, 通过每次将点集划分为两个子集, 递归地对两个子集进行 Delaunay 三角剖分, 然后将两个子集的剖分结果合并起来. 运用一些巧妙的技巧可以使得合并操作的复杂度为 $O(n)$, 于是算法的复杂度为 $O(n \log n)$, 并可以进一步优化为 $O(n \log \log n)$.

1.4.3 Poisson 表面重建

与 Delaunay 三角剖分这种由点云直接构造三角网格的方法相比, Poisson 表面重建则采取了相对间接的方法, 先构建符号距离函数 (Signed Distance Function, SDF), 然后通过提取等值面来获得表面模型. 这可以避免对点云直接建模时可能出现的孔洞和噪声问题.

定义 1.9 Poisson 表面重建 **Poisson 表面重建 (Poisson Surface Reconstruction)** 是一种基于点云法线信息, 通过求解 Poisson 方程来重建连续表面模型的方法.

具体而言[3][4], 对于空间中的几何体 M, 定义函数 χ_M 使得

$$\chi_M(\boldsymbol{v}) = \begin{cases} 1, & \boldsymbol{v} \in M \\ 0, & \boldsymbol{v} \notin M \end{cases}$$

该函数称为 M 的指示函数 (Indicator Function). 注意到 χ_M 在 M 的表面 ∂M 处是不连续的, 为了将其与表面的梯度场联系起来, 我们可以对其进行平滑处理, 即对 χ_M 与高斯函数进行卷积操作:

$$\left(\chi_M \circ \tilde{F}\right)(\boldsymbol{v}) = \int \tilde{F}\left(\boldsymbol{v}-\boldsymbol{u}\right)\chi_M(\boldsymbol{u})\mathrm{d}\boldsymbol{u} = \int_M \tilde{F}_{\boldsymbol{v}}(\boldsymbol{u})\mathrm{d}\boldsymbol{u}, \quad \tilde{F}_{\boldsymbol{v}}(\boldsymbol{u}) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{||\boldsymbol{u}-\boldsymbol{v}||^2}{2\sigma^2}\right)$$

其中已经带入了 χ_M 的定义. 以下为了方便考虑, 重新定义 $\chi_M = \chi_M \circ \tilde{F}$. 通过散度定理可以证明:

$$\nabla\chi_M(\boldsymbol{v}) = \int_{\partial M} \tilde{F}_{\boldsymbol{v}}(\boldsymbol{u})\boldsymbol{n}_{\boldsymbol{u}}\mathrm{d}\boldsymbol{u}$$

其中 $\boldsymbol{n}_{\boldsymbol{u}}$ 是 ∂M 在点 $\boldsymbol{u}$ 处的内法向量[5].

现在, 我们需要将连续的梯度场 $\nabla\chi_M$ 与离散的, 带法向量的点云 S 联系起来. 首先, 需要根据带法向量的点云 S 重建一个向量场 $\boldsymbol{V}$ 用于之后的最小化运算. 根据 S 中的点 $\boldsymbol{v}_i$ 将曲面划分为不相交的局部区域 $\mathscr{P}_i$, 于是对于空间中任意一点 $\boldsymbol{v}$ 有

$$\nabla\chi_M(\boldsymbol{v}) = \sum_{\boldsymbol{v}_i \in S}\int_{\mathscr{P}_i} \tilde{F}_{\boldsymbol{v}}(\boldsymbol{u})\boldsymbol{n}_{\boldsymbol{u}}\mathrm{d}\boldsymbol{u}$$

在每个 $\mathscr{P}_i$ 内, 我们近似地用 $\boldsymbol{v}_i$ 代替式中的 $\boldsymbol{u}$, 从而得到

$$\nabla\chi_M(\boldsymbol{v}) \approx \sum_{\boldsymbol{v}_i \in S}\int_{\mathscr{P}_i} \tilde{F}_{\boldsymbol{v}}\left(\boldsymbol{v}_i\right)\boldsymbol{n}_i\mathrm{d}\boldsymbol{v}_i = \sum_{\boldsymbol{v}_i \in S}\tilde{F}_{\boldsymbol{v}}\left(\boldsymbol{v}_i\right)\boldsymbol{n}_i\int_{\mathscr{P}_i}\mathrm{d}\boldsymbol{v}_i = \sum_{\boldsymbol{v}_i \in S}\tilde{F}_{\boldsymbol{v}}\left(\boldsymbol{v}_i\right)\boldsymbol{n}_i\left|\mathscr{P}_i\right| \xlongequal{\text{定义为}} \boldsymbol{V}(\boldsymbol{v})$$

其中 $|\mathscr{P}_i|$ 为该邻域的面积. 于是重建的向量场 $\boldsymbol{V}(\boldsymbol{v})$ 可以通过点云加权平均得到.

现在的目标是根据重建的向量场 $\boldsymbol{V}$ 来求解指示函数 χ_M. 根据上式, 我们需要解如下的梯度场方程:

$$\nabla\chi_M = \boldsymbol{V}$$

[3]Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In Proceedings of the Fourth Eurographics Symposium on Geometry Processing, SGP '06, 61-70. Goslar, DEU, 2006. Eurographics Association.

[4]文献作者总是采用一些花哨的符号以展现他的数学功底, 可惜这对读者们造成了很大困扰.

[5]这里取法向量的方法与一般的习惯不同, 需要加以注意.

由于右侧的 $\boldsymbol{V}$ 不一定可积, 因此可能无法找到方程的精确解. 于是我们转而求解下面的最小化问题:

$$\arg\min_{\chi_M} = \int_M ||\nabla\chi_M - \boldsymbol{V}||^2 \,\mathrm{d}\boldsymbol{v}$$

这一最小化问题的解为:

$$\Delta\chi_M = \nabla \cdot \boldsymbol{V}$$

通过最小二乘法求解该方程, 即可得到 χ_M 的近似解. 最后, 通过 Marching Cubes 等等值面提取算法, 从 χ_M 中提取出表面模型.

Poisson 表面重建依赖于带法向量的点云 $\boldsymbol{V}$ 的构建. 由于点云在空间上不是均布的, 我们需要高效的对其离散化和处理. 常用的办法是八叉树 (Octree) 结构, 它能够自适应地对点云进行空间划分, 在点云密集的区域使用更细的划分, 而在点云稀疏的区域使用较粗的划分. 这样不仅节省了存储空间, 也提高了计算效率. 八叉树的构造方法如下:

1. 初始化一个包含所有点的立方体区域作为八叉树的根节点.

2. 递归地将每个节点划分为八个子节点, 直到满足以下条件之一:

- 节点内的点数小于预设的阈值.
- 达到预设的最大深度.

3. 对于每个叶节点, 计算其包含的点的法向量信息, 并将其存储在节点中.

在八叉树的每个节点上定义局部基函数 (通常是线性函数)$\phi_i(\boldsymbol{v})$, 那么上述问题即可转化为线性方程组的求解问题, 从而得到最终的 χ_M.

1.5 隐式表面重建——Marching Cubes 算法

符号距离场函数 f 的定义如下:

$$f_M(\boldsymbol{v}) = \begin{cases} \min\limits_{\boldsymbol{v}'\in\partial M} ||\boldsymbol{v}-\boldsymbol{v}'||, & \boldsymbol{v}\in M \\ -\min\limits_{\boldsymbol{v}'\in\partial M} ||\boldsymbol{v}-\boldsymbol{v}'||, & \boldsymbol{v}\notin M \end{cases}$$

它本身隐式地表示了几何形状, 其 0-等值面对应着几何形状的表面. 许多方法能够通过对符号距离场求值来计算几何表面的位置. 行进立方体 (Marching Cubes) 算法就是这样一种方法, 它能够从符号距离函数中重建网格模型, 被广泛使用于整个几何建模领域.

Marching Cubes 算法的基本思想是将三维空间划分为一系列立方体单元, 然后在每个立方体内根据符号距离函数的值来确定表面与立方体的交点. 具体步骤如下:

1. 将三维空间划分为均匀的立方体网格.

2. 对于每个立方体的所有顶点 $\boldsymbol{v}_i(i=0,\cdots,7)$, 其 SDF 值 $sdf\,(\boldsymbol{v}_i)$ 的正负值决定了其在几何体的外部或内部. 如果一条边的两个顶点的 SDF 值符号不同, 则表明该边与几何体的表面相交. 如此, 每个立方体的顶点的符号组合共有 $2^8 = 256$ 种可能, 可以将其编码为一个 8 位的二进制数 c. Marching Cubes 算法预先定义了两张查找表: 一张是边状态表 $\mathcal{E}$, 用于指示哪些边与表面相交; 另一张是三角形表 $\mathcal{T}$, 用于指示如何连接这些交点以形成三角形网格.

3. 首先, 根据立方体顶点的符号组合计算出索引 c. 然后, 查找边状态表 $\mathcal{E}[c]$ 以确定哪些边与表面相交. 对于每条相交的边, 通过线性插值计算出交点的位置, 即

$$\boldsymbol{v}^* = \frac{v_2 - v^*}{v_2 - v_1}\boldsymbol{v}_1 + \frac{v^* - v_1}{v_2 - v_1}\boldsymbol{v}_2$$

其中 $\boldsymbol{v}^*$ 是交点位置, $\boldsymbol{v}_1, \boldsymbol{v}_2$ 是边的两个端点, v_1, v_2 是顶点的 SDF 值, $v^* = 0$ 表示表面的 SDF 值.

然后, 根据三角形表 $\mathcal{T}[c]$ 确定需要将哪些交点连接成三角形, 将交点位置和生成的三角形添加到最终的网格模型中.

4. 重复步骤 2 和 3, 直到处理完所有立方体单元.

1.6 点云的模型拟合

模型拟合指的是从给定的点云中提取出平面或球体, 圆柱体等几何基元的过程. 它在低层次的三维数据 (无序的点集) 和高层次的三维形状的结构信息之间建立了联系, 为许多下游的应用提供基础.

我们可以如下定义模型拟合问题[6]: 对于给定的点云 $P = \{\boldsymbol{v}_i : i = 1, \cdots, N\}$ 和给定的一类几何形状 S, 求 P 的子集 P' 和 S 的方程 $F(\boldsymbol{v})$ 的参数, 使得 $\forall \boldsymbol{v}_j \in P', F(\boldsymbol{v}_j) = 0$. 由于实际的点云有一定的噪声, 因此通常不能做到完美拟合. 我们一般定义一个误差函数, 然后通过最小二乘法等数值方法近似求解.

1.6.1 拟合方法示例——平面拟合

我们考虑一个比较简单的例子: 输入的点云 P 近似在一个平面 $ax + by + cz + d = 0$ 上. 定义误差函数 E 为点到平面的距离的平方和:

$$E = \sum_{i=1}^{N} d_{\boldsymbol{v}_i} = \sum_{i=1}^{N} (ax_i + by_i + cz_i + d)^2$$

写成矩阵乘法的形式即

$$E = ||\boldsymbol{P}\boldsymbol{M}||^2, \quad \boldsymbol{P} = \begin{bmatrix} \boldsymbol{v}_1^{\mathrm{t}} & 1 \\ \vdots & \vdots \\ \boldsymbol{v}_N^{\mathrm{t}} & 1 \end{bmatrix}, \quad \boldsymbol{M} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

对 $\boldsymbol{P}$ 奇异值分解, 有

$$\boldsymbol{P} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{t}}$$

由于 $E = \left|\left|\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{t}}\boldsymbol{M}\right|\right|^2 = \left|\left|\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{t}}\boldsymbol{M}\right|\right|^2$(正交矩阵不改变向量的模长), 因此为了最小化 E, 需要 $\boldsymbol{M}$ 应当是 $\boldsymbol{V}^{\mathrm{t}}$ 的最后一列 (这样才能对应于最小的奇异值). 这样就得到了平面的参数 a, b, c, d.

[6]对二维空间中的数据进行线性拟合实际上就可以视作对二维点云的直线拟合.

1.6.2 随机抽样一致性 (RANSAC) 算法

实际问题是复杂的, 因为 P 中的某些点并不属于待拟合的几何形状[7]. 为了解决这个问题, 我们可以使用随机抽样一致性 (RANSAC, Random Sample Consensus) 算法.

对于点云 P 中的点和待求的形状 S, 显然有相当一部分点落在 S 上 (在实际情况中也可能是 S 附近的一个区域, 下文的落在 S 上也即这种含义). 仍然以平面的拟合为例, 如果落在 S 上的概率为 w, 那么从点云中随机选择 n 个点, 恰好确定目标平面的概率为 w^n. 重复这一过程 k 次, 成功使得取样点均在 S 上的概率为

$$p = 1 - (1 - w^n)^k$$

RANSAC 算法的基本思想正是通过多次随机采样来增加找到正确模型的概率. 具体步骤如下:

1. 随机从点云 P 中选择一定数量的点 (通常选取的较少, 以免需要的采样次数过多. 对于平面这种简单的图形, 一般选取三个点即可) 来确定一个候选模型 S'.
2. 计算点云 P 中每个点到模型 S' 的距离, 并将距离小于预设阈值的点视为内点 (即属于模型的点). 统计内点的数量.
3. 重复步骤 1 和 2 多次, 记录内点数量最多的模型 S^* 及其对应的内点集 P^*.
4. 使用内点集 P^* 重新拟合模型, 得到最终的几何形状.

[7] 例如一个建筑物的点云中不可能所有点都属于某一特定的墙面, 除非这个建筑物是一堵墙.