# 基于 Voronoi Diagram 的车道中心线生成

## I. 问题

传统的车道检测方法通常在光线不足、遮挡和车道磨损等各种条件下难以实现，导致结果不可靠、断断续续。为了解决这些问题，本项目采用 Voronoi Diagram 来精确生成车道中心线。主要目标是实现数据的有效预处理与后处理，利用 Voronoi Diagram 进行精确的中心线计算，并优化算法以确保实时计算的效率。

## II. 实现

### 1. 预处理

#### i. DBSCAN Clustering

利用 DBSCAN 聚类算法将乱序的数据点分类为不同的车道线。DBSCAN 的主要优势在于它能有效识别和处理任意形状的聚类，并且对噪声点具有较好的鲁棒性。该算法通过测量数据点的局部密度来确定是否将其包含在某个聚类中，从而允许它适应复杂的车道线形状。

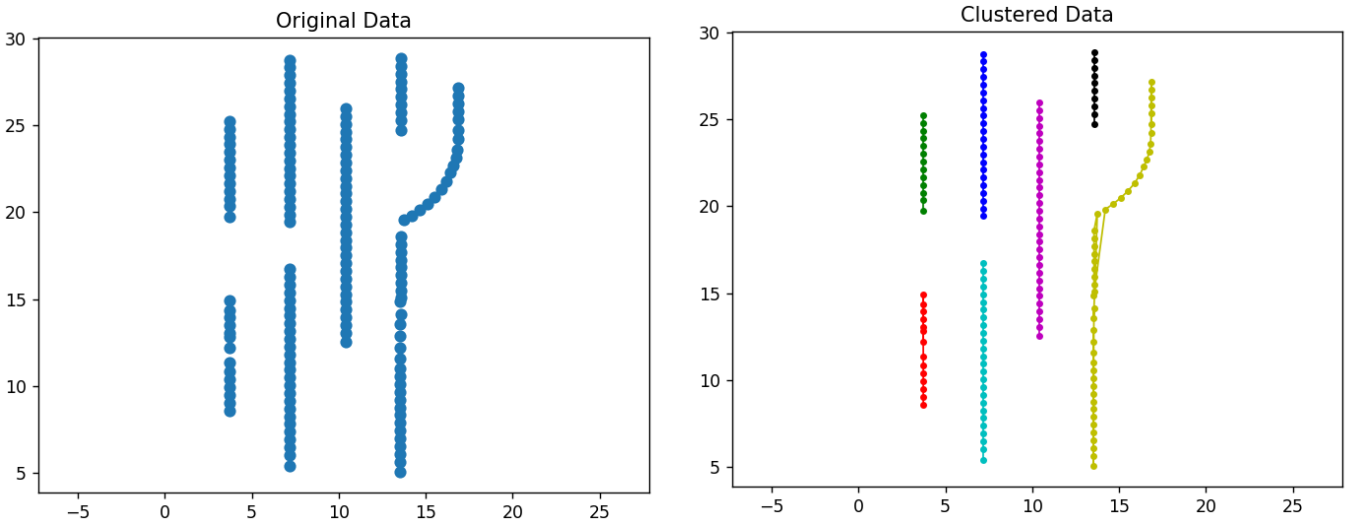



*Figure 1: DBSCAN 效果对比。*

ii. merge_line_on_side() - 可选

该功能通过将靠近图像边缘的断裂车道线合并，来减少由于车道线检测不准确造成的错误。这种方法提高了车道线识别的连续性和完整性，尤其是在道路边缘条件复杂或车道线部分被遮挡的情况下。

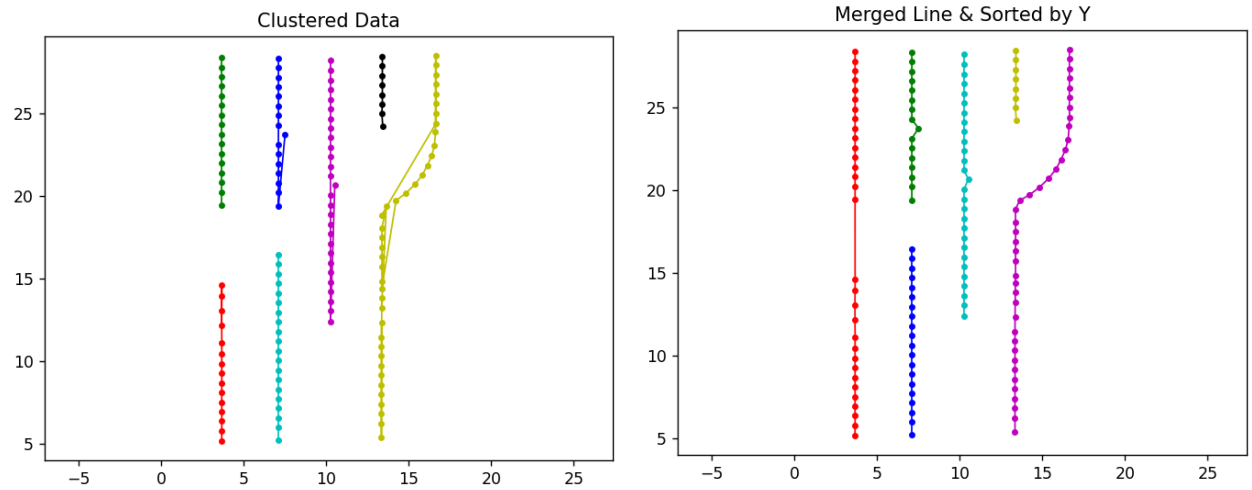



*Figure 2: 合并边缘线并排序效果前后对比。*

iii. 基于 B-Spline 的车道线平滑 - 可选

使用 B-Spline 插值技术对识别出的车道线进行平滑处理，有效地减少噪点和不规则数据点的影响。B-Spline 方法提供了高度的灵活性和平滑度，能够生成更加符合实际车道线曲率的平滑曲线。

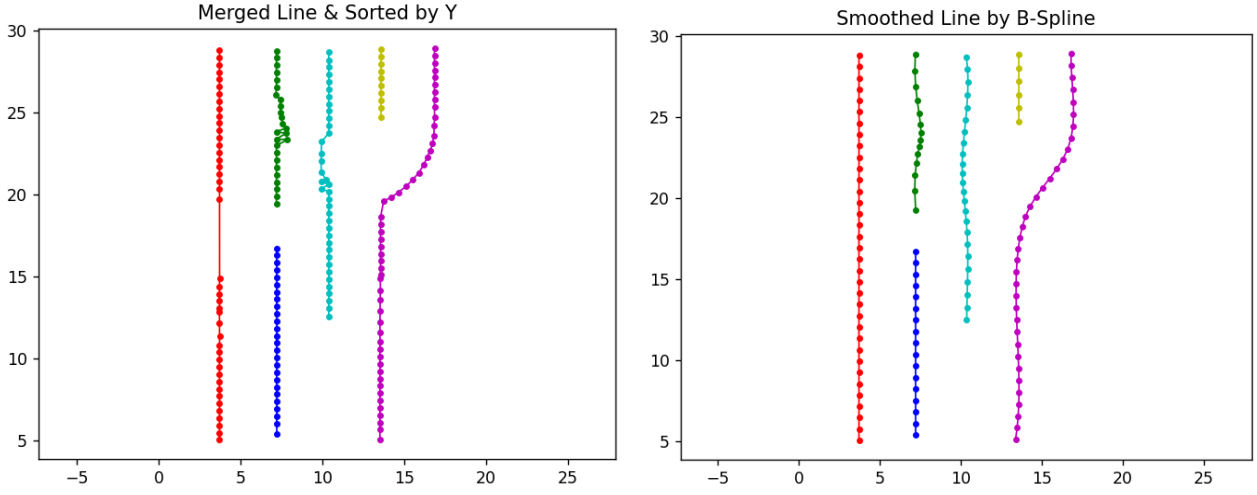



*Figure 3: 基于 B-Spline 插值的车道线平滑前后效果对比。*

iv. 基于多项式回归的车道线平滑 - 可选

使用多项式回归拟合出最适合车道线数据点的曲线模型，再进行重采样以达到车道线平滑/去除杂点的效果，提高数据质量。

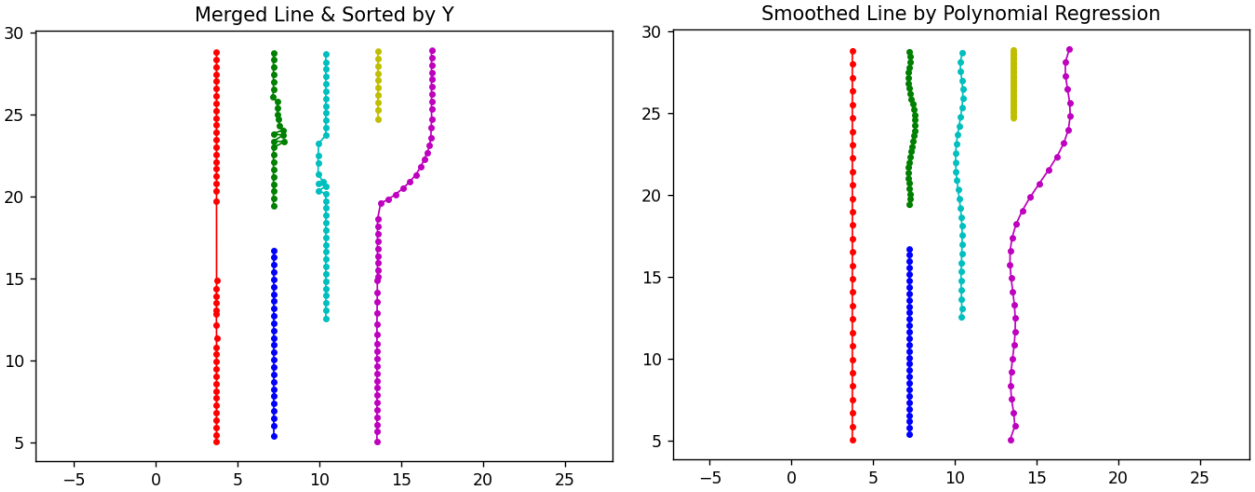



*Figure 4: 基于多项式回归的车道线平滑效果对比。*

v. Moving Average Filter - 可选

通过移动平均滤波器处理车道线数据，可以有效地消除数据中的随机噪声。该滤波器通过计算数据点的移动平均值，平滑车道线的整体表示，从而增强车道线的可识别性和稳定性。

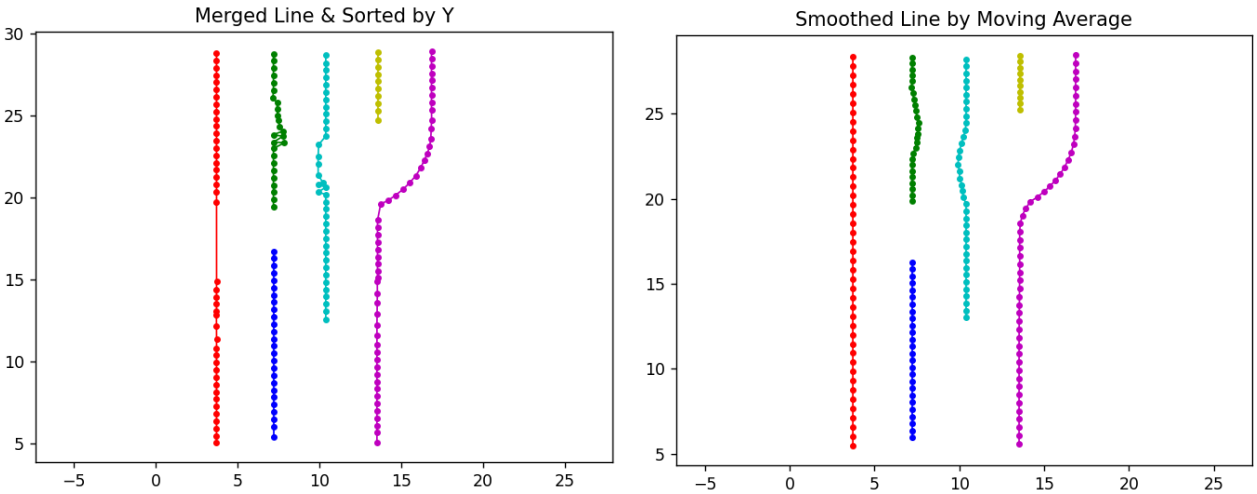



*Figure 5: 滑动平均效果前后对比。*

vi. Alpha Shape

使用 Alpha Shape 算法确定能够包围整个图像区域的最小多边形边界，用于识别和清除 Voronoi 图在车道线外围可能生成的误判点。该方法不仅增强了车道线的识别准确性，也为后续的车道线跟踪和识别提供了更清晰的边界参考。

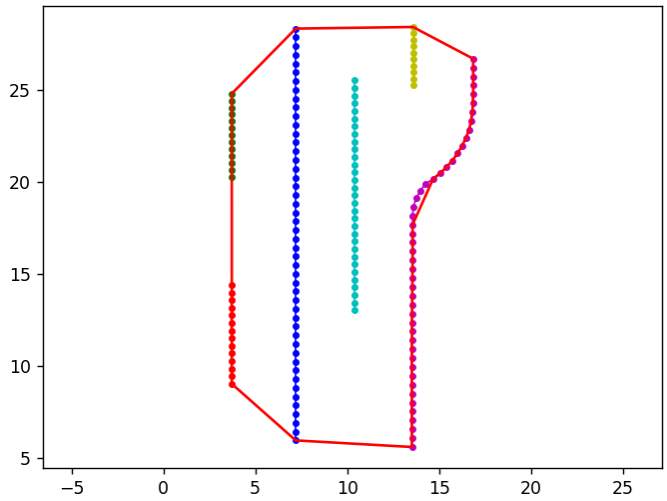



*Figure 6: 通过 Alpha Shape 找到多边形外包围线(红色)，其它为车道线。*

（因为后来给我的数据是已经处理过的车道线，所以预处理的 i 到 v 不需要了）

2. 生成 Voronoi Diagram

使用 jcv_voronoi 库进行生成，其基于 Fortune's sweep algorithm. 此库比多数开源库运行更快，且更为简单可靠，内存占用少。

https://github.com/JCash/voronoi

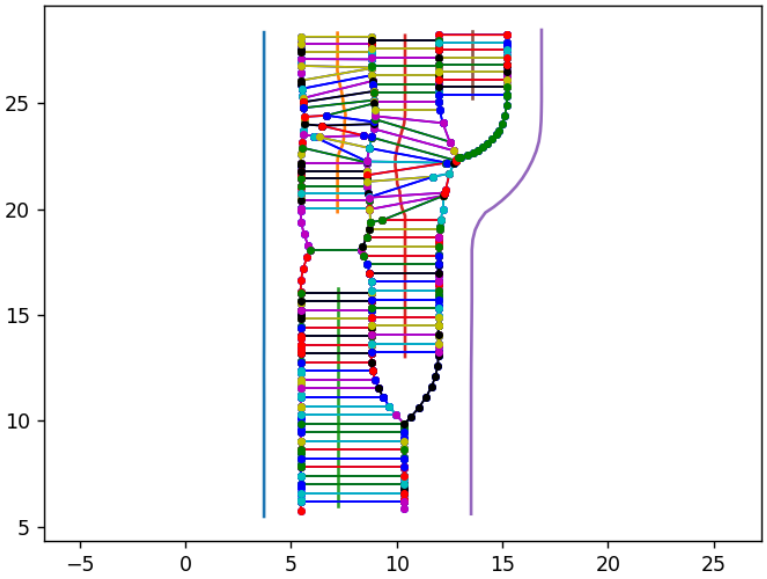


*Figure 7: 只处理了车道外线段的 Voronoi Diagram。*

通过维诺图生成车道中心线的方法是基于维诺图对路面进行空间划分，通过查找相邻左右车道线点区域的共享边来确定中心线，其代表着两个不同边界点的等距线，理论上位于车道中央，提取出这些线段(共享边)就可以形成平滑的中心线。

3. 后处理

Voronoi Diagram 生成出来的结果是很多条小线段，包含众多杂点。因此需要对其进行后处理以删除多余部分。

根据 Voronoi Diagram 的理论。只需要通过寻找不在车道线的外包围多边形的线段，我们就可以找出在车道外的杂点。然后寻找与每一根车道线相交的线段并清除掉。剩下的部分便是干净的中心线（如果初始数据很杂乱且车道线不进行平滑处理的话还是会留下一些小杂点），且相邻线段间一定是相连的。

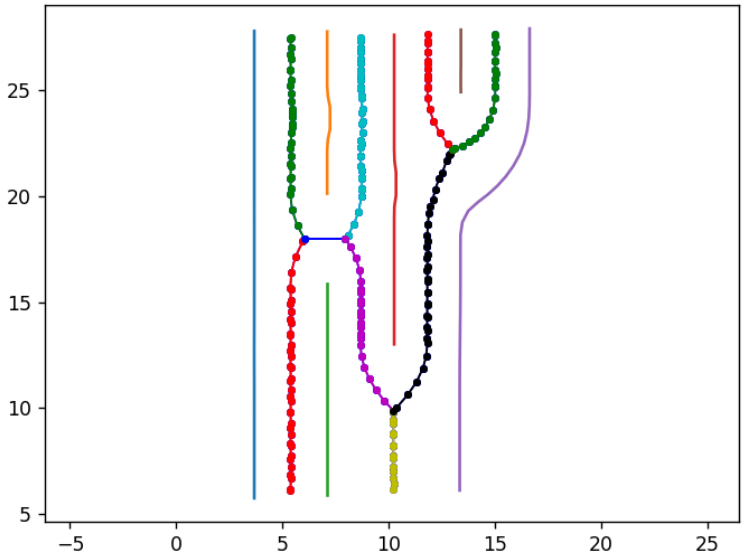



*Figure 8: 剔除多余线段后干净的中心线，并分辨出中心线的每个分支*

基于这个需求我选择了 r-tree 数据结构，r-tree 是空间数据索引的方法，可以快速的基于各个对象的空间关系进行查询(query)和处理。

然后对中心线进行合并处理，若一个节点只连到一条边则当作端点，节点连到两条边则当作组成线的点之一，节点连到三条边或以上则当作分支点。合并后的中心线如 figure 8 所示，可以知道哪些点属于哪些线还有线与线之间的拓扑关系。

i. 优化 1

在处理车道线识别中，我面临着如何高效筛选出有效线段的挑战。具体来说，我们首先从 R-tree 中提取数据，结果被分为两个向量：一个包含位

于外围多边形内的索引（需要保留的车道线段），另一个则是与车道线相交的索引（需要删除的线段）。这里的索引指的是 Voronoi 图中生成的所有小线段的向量索引。传统的处理方法通过逐一遍历或查找来确定哪些索引同时存在于两个向量中，这种方法虽直观但效率低下。

为了解决这一效率问题，我提出了一个创新的方法：创建两个主向量长度的二进制数，每个索引位置用 0 或 1 表示其是否应被保留（1 表示保留，0 表示删除）。通过对这两个二进制数进行逻辑“与”操作(AND gate)，即可快速得到最终需要保留的线段的索引。这种方法不仅大幅提高了运算速度，还显著减少了内存占用，优化了整个车道线处理流程的性能。

ii. 优化 2

在车道线平滑度不足的情况下，即使经过后处理，生成的中心线仍可能包含杂乱无章的线条。为了解决这一问题，我引入了一种剔除方法，专门移除那些过于靠近车道线的中心线。这不仅有助于清理生成的中心线，保证其清晰度和可用性，而且增强了中心线的可靠性。

此外，这种方法还具有辅助作用，能帮助我们评估车道的宽窄是否符合安全通过的标准。如果某条道过窄，通过此方法我们可以及时识别并采取相应措施，确保驾驶安全。通过这种优化，我们不仅提高了中心线的生成质量，也为车道安全评估提供了技术支持。

iii. 优化 3

经过性能评估，我发现对 r-tree 进行查询以筛除边界外或过于靠近车道线的维诺图输出这个步骤运行缓慢，占用了运行时间中的大部分。经过分析，我发现这是因为使用 r-tree 对几何图形间关系的查询会对整个 r-tree 中的几何图形进行匹配导致效率不佳，特别是边缘复杂顶点多的图形运行速度会特别慢。

经过研究，此问题可通过网格分割的方法解决。将整个大图形分割成多个正方形的小块放入 r-tree 中，然后查询维诺图输出的线段与 r-tree 中图形的相交或距离关系。如图 9 所示，一个复杂图形分割为多个子多边形，第一轮查询中找出目标所在的子多边形，然后用这个子多边形与目标进行相交或距离关系的查询。此方法可省去大部分的计算以提高运行效率。

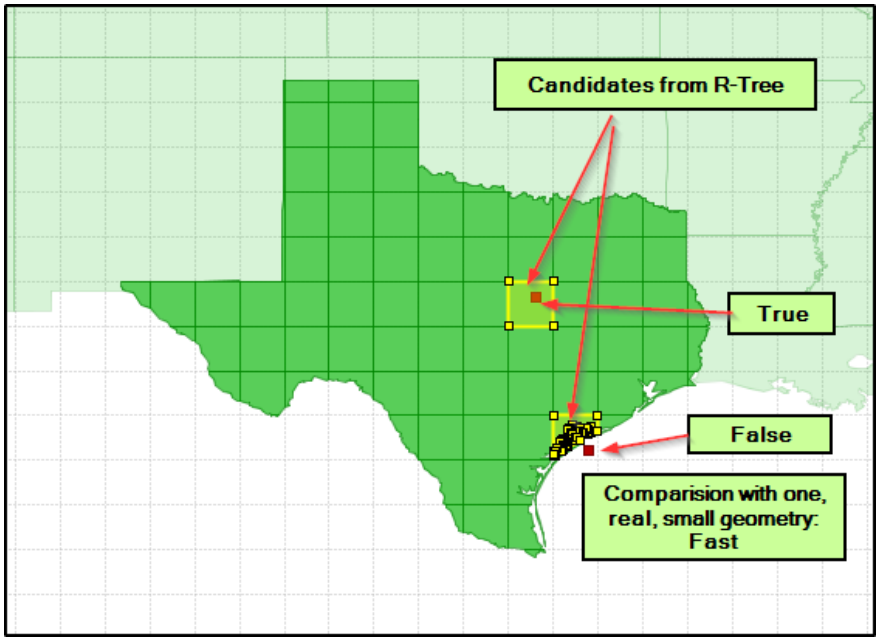
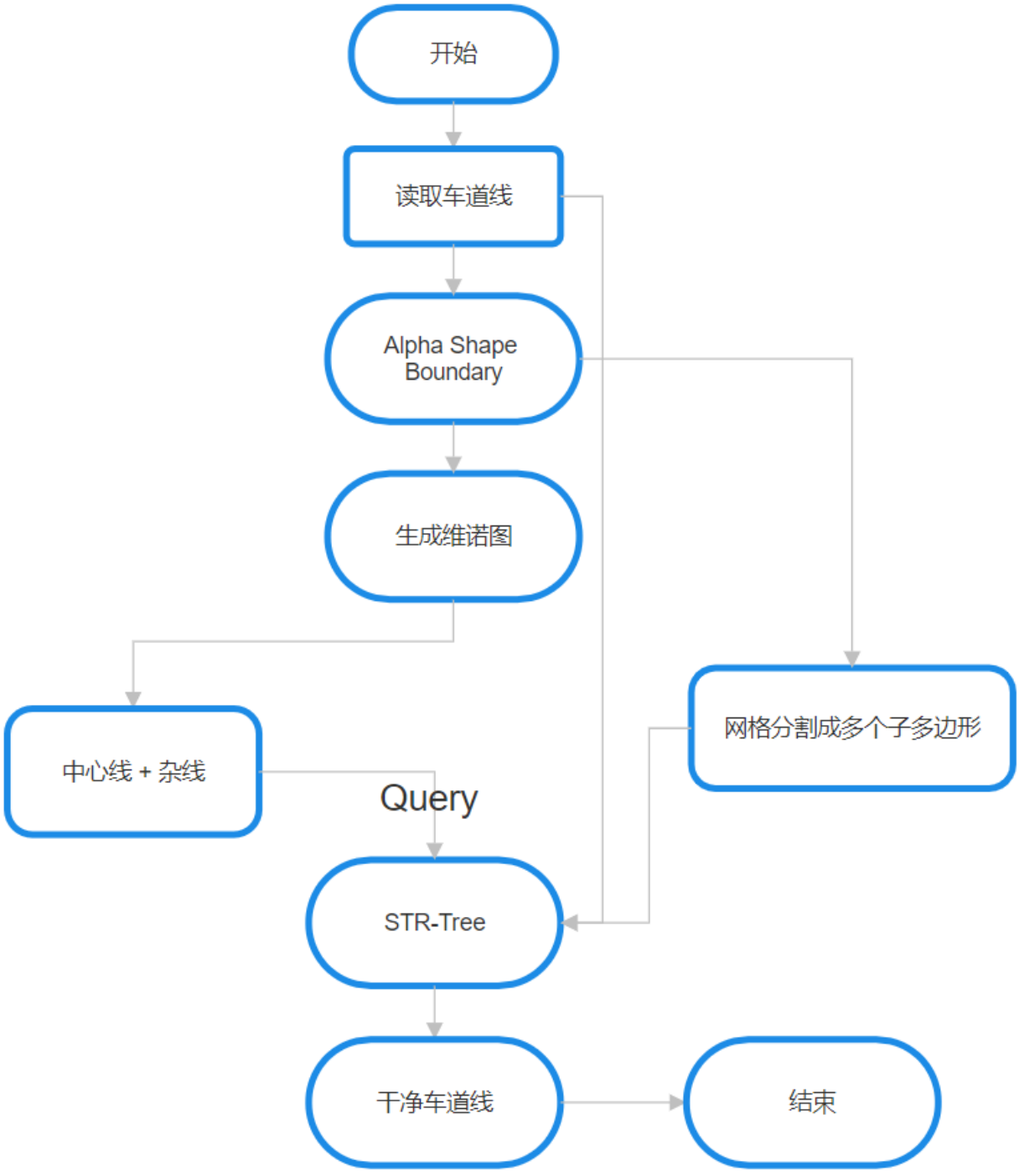
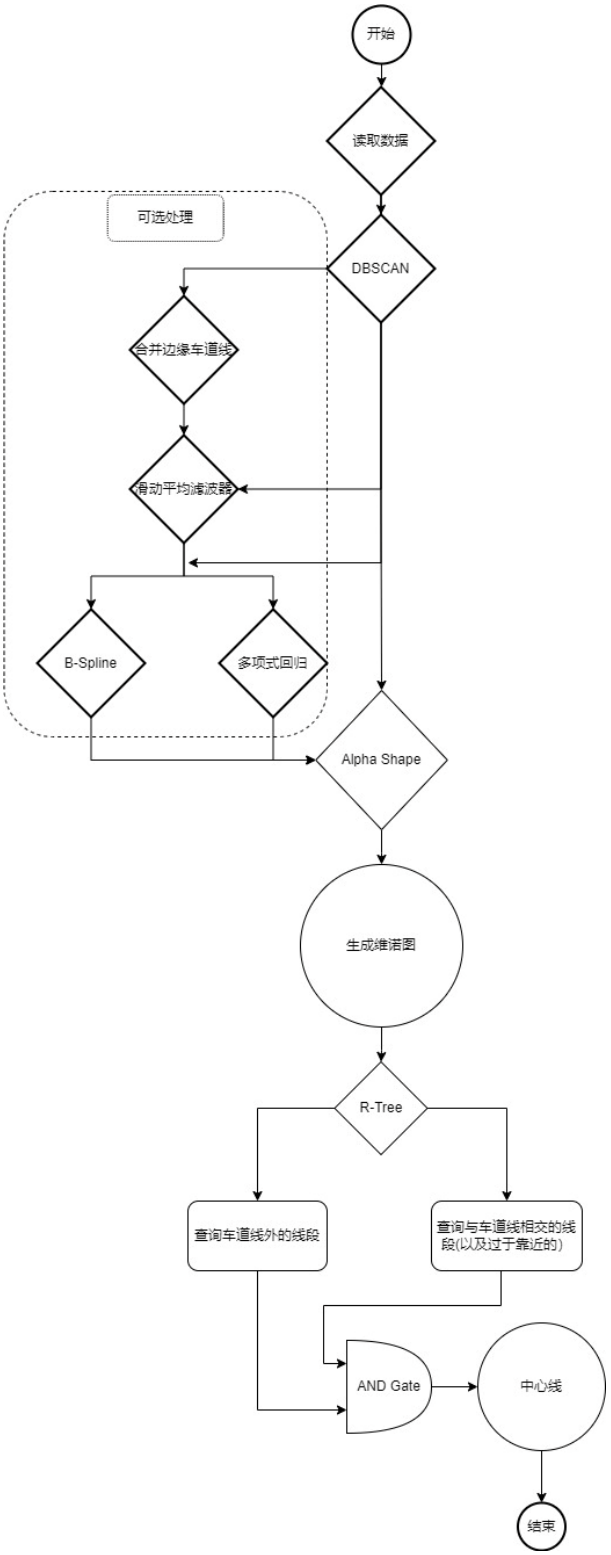



*Figure 9: 通过网格分割加速 r-tree 查询策略。*

Pipeline（新）：


开始
读取车道线
Alpha Shape Boundary
生成维诺图
中心线 + 杂线
网格分割成多个子多边形
Query
STR-Tree
干净车道线
结束

Pipeline（旧）：


开始
读取数据
可选处理
DBSCAN
合并边缘车道线
滑动平均滤波器
B-Spline
多项式回归
Alpha Shape
生成维诺图
R-Tree
查询车道线外的线段
查询与车道线相交的线段(以及过于靠近的)
AND Gate
中心线
结束