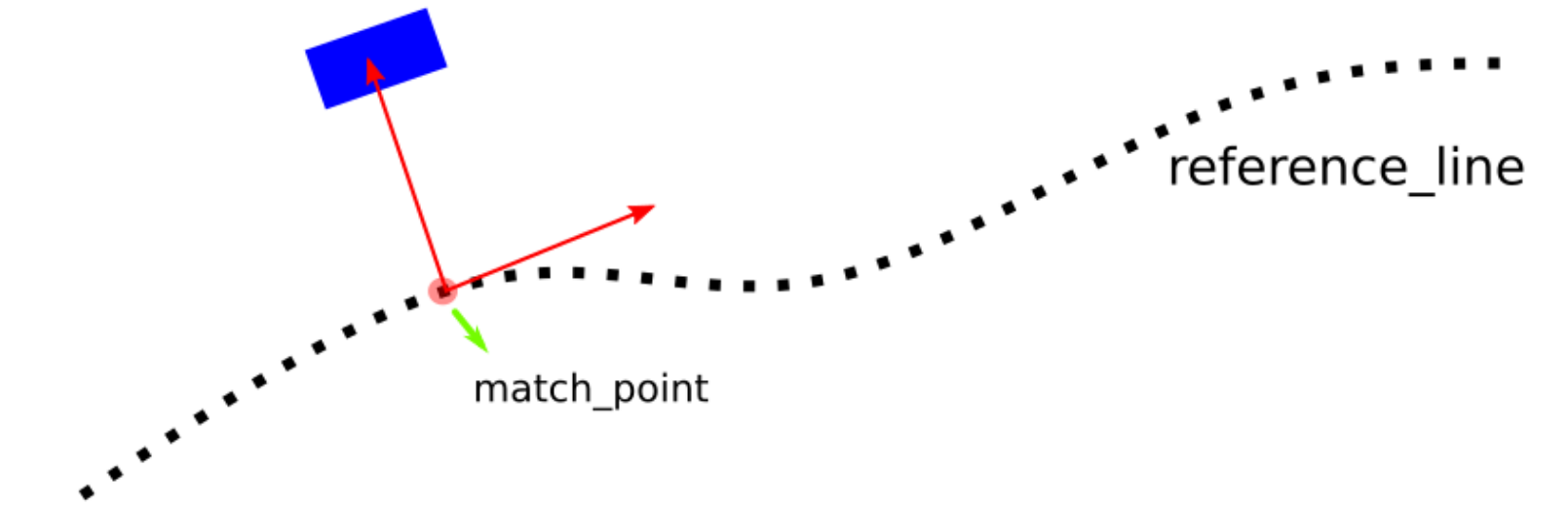
问题

如何在traj上当前位置在reference_line上的match_point(匹配点)?

匹配点定义

位于reference_line上，且在该点处的切线与车辆当前位置和匹配点之间的连线相互垂直

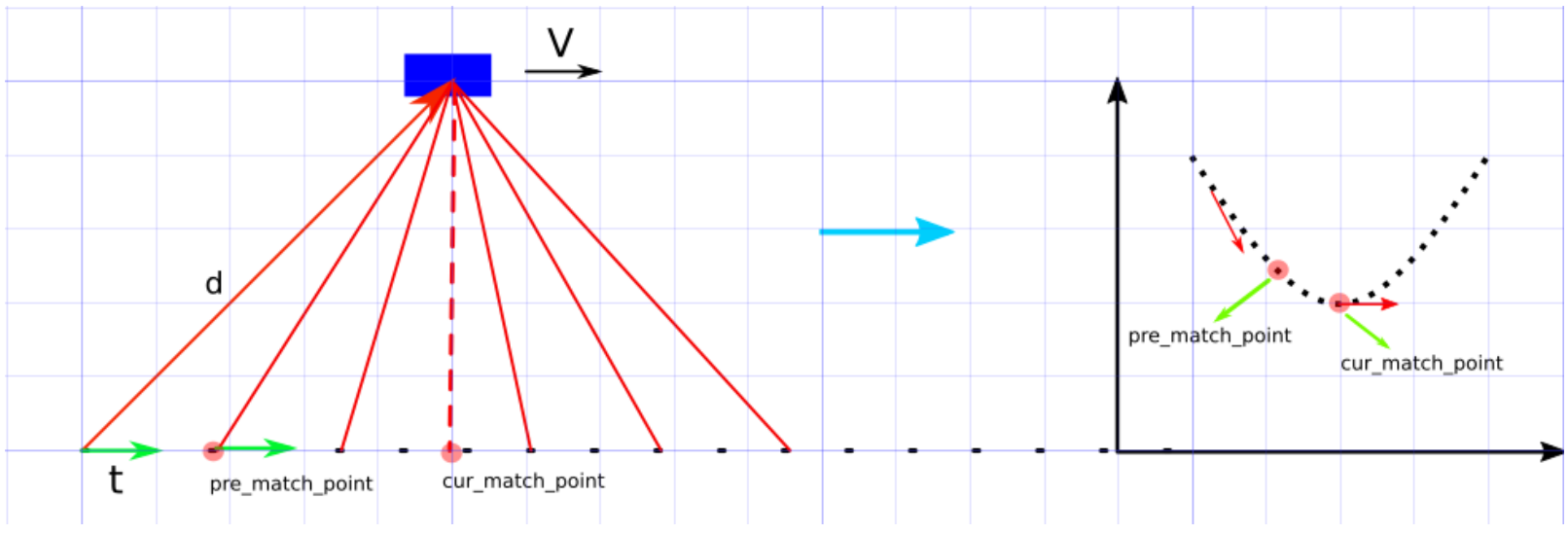


改进搜索匹配点方法

思路:

- 同一参考线上，利用前一个匹配点的索引作为下一个搜索匹配点循环的起始点
- 计算 $\vec{d}\cdot\vec{t}$,用来判断遍历的方向(车向前开或者向后开)
- 利用变量 `increase_count` 记录距离增加的次数，以减少向后搜索的次数
  - `abs(dis) < epsilon`,则认为当前匹配点和前一个匹配点重合

```
pre_index ← 0
cur_index ← 0
epsilon ← 1e-3

for t ← pre_index to refrencelinrsize()
  if abs(dis) < epsilon
    then cur_index ← pre_index

  if dis[i+1] < dis[i]
    then increase_count ← 0
  else
    then increase_count ← increase_count + 1

  if increase_count > 10
    then break
```

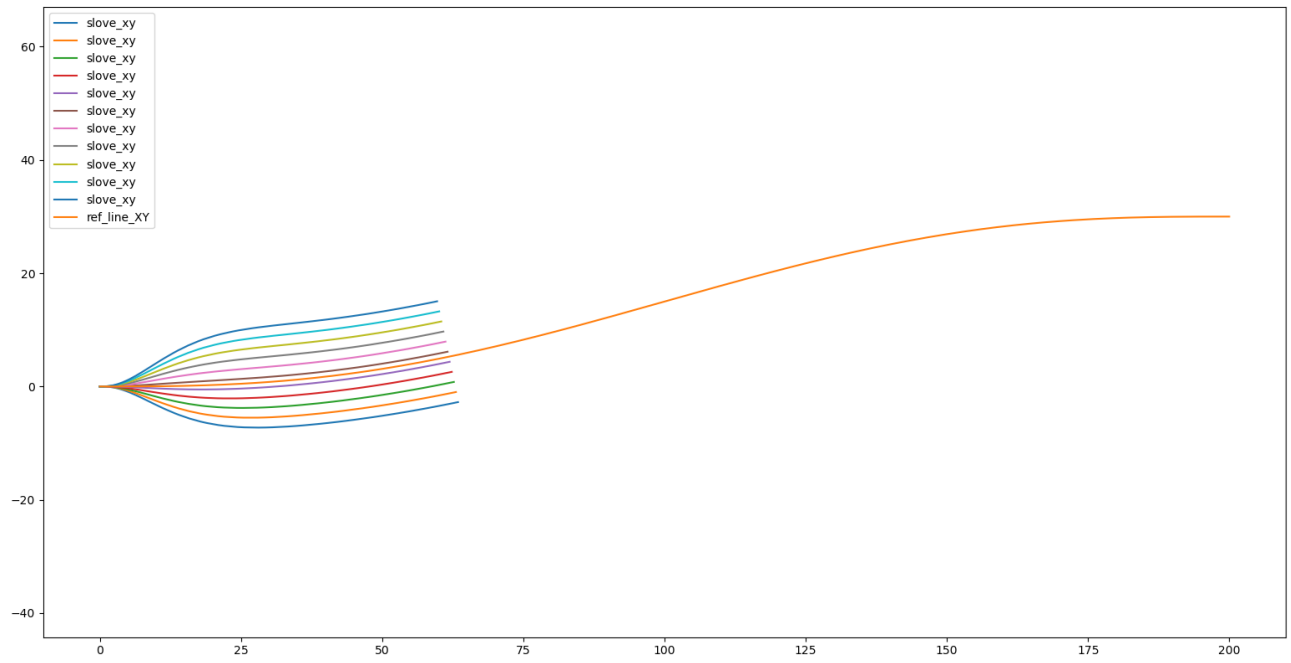



```
void QuinticPolynomial::matchPoint(const FrenetPath fp,
                                   const double current_post_x,
                                   const double current_post_y, int pre_index,
                                   int& index) {
  //计算上一个的匹配点的位矢
  std::array<double, 2> pre_cur_error{current_post_x - fp.x.at(pre_index),
                                      current_post_y - fp.y.at(pre_index)};
  double heading = fp.theta.at(pre_index);
```

```
    std::array<double, 2> pre_heading{cos(heading), sin(heading)};

    double innerProd = pre_cur_error.at(0) * pre_heading.at(0) +
                       pre_cur_error.at(1) * pre_heading.at(1);
    // TODO: 车往前或者往后开
    // if (innerProd > 0) {}

    double epsilon = 1e-3;

    size_t numPoints = fp.x.size();
    int increase_count = 0;
    double dis_min = std::numeric_limits<double>::max();
    for (size_t i = pre_index; i < numPoints; ++i) {
      if (std::abs(innerProd) < epsilon) {
        index = pre_index;
        break;
      }
      double temp_dis = std::pow(fp.x[i] - current_post_x, 2) +
                        std::pow(fp.y[i] - current_post_y, 2);

      if (temp_dis < dis_min) {
        dis_min = temp_dis;
        index = i;
        increase_count = 0;
      } else {
        ++increase_count;
      }

      if (increase_count > 10) {
        break;
      }
    }
  }


```

搜索时间 (200个点)

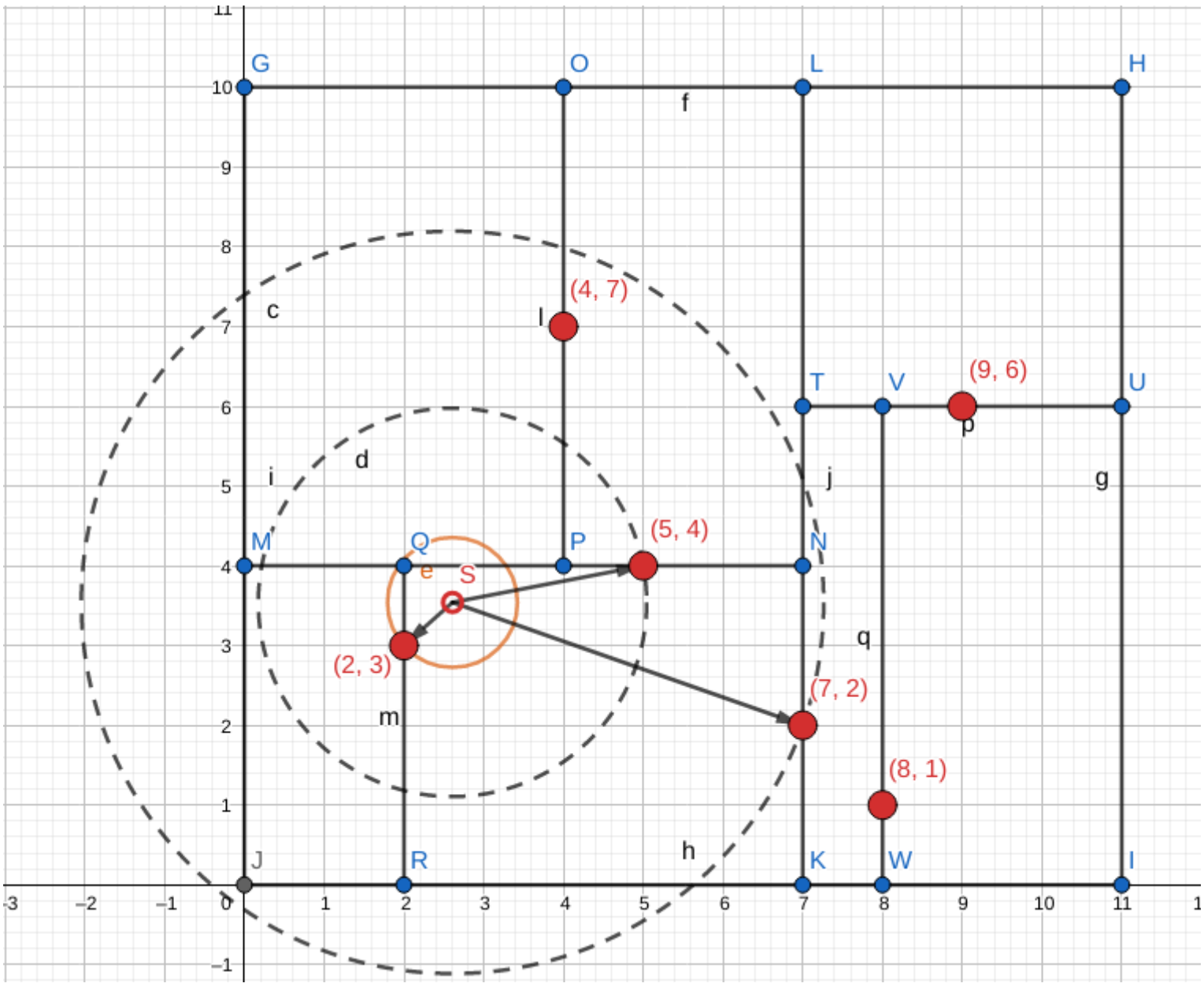
```
①遍历整条参考线
  Time for matching Point = 0.017 msec.
②以前一个匹配点作为寻找当前匹配点循环的起始点
  Time for matching Point = 0.008 msec.
③添加increase_count作为终止条件
  Time for matching Point = 0.004 msec.
```



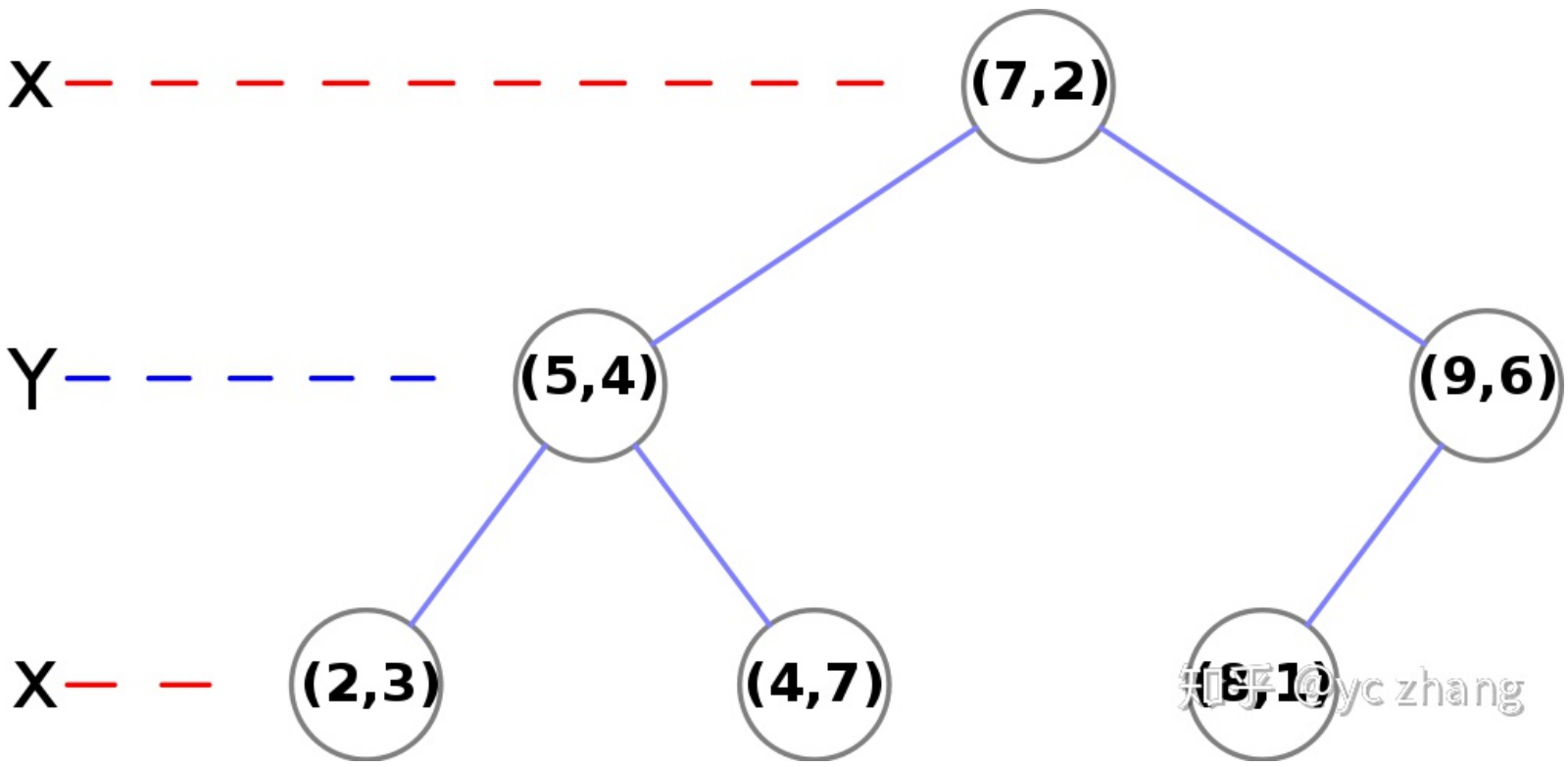
kd-tree

原理

二维样例：{（2,3），（5,4），（9,6），（4,7），（8,1），（7,2）}



树形结构



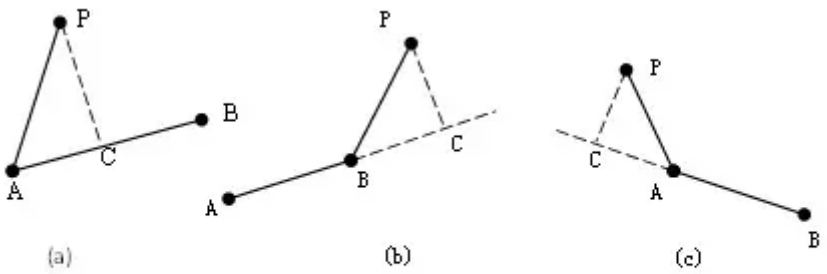
搜索效率对比

```
①遍历整条参考线
  Time for matching Point = 0.017 msec.
②以前一个匹配点作为寻找当前匹配点循环的起始点
  Time for matching Point = 0.008 msec.
③添加increase_count作为终止条件
  Time for matching Point = 0.004 msec.
④使用kd-tree搜索
  Time for matching Point = 0.0017 msec.
  初始化kd-tree耗时: 0.1 msec

上述方法搜索200个匹配点总耗时:
 ① 0.017 * 200 = 3.4 msec
 ③ 0.004 * 200 = 0.8 msec
 ④ 0.1 + 0.0017 * 200 = 0.44 mesc
```

出现的问题

```
min_index_y:23 ,min_index_n:24
min_index_y:0 ,min_index_n:0
min_index_y:99 ,min_index_n:99
min_index_y:0 ,min_index_n:0
min_index_y:99 ,min_index_n:99
min_index_y:99 ,min_index_n:99
min_index_y:0 ,min_index_n:0
min_index_y:0 ,min_index_n:0
min_index_y:11 ,min_index_n:12
min_index_y:65 ,min_index_n:65
min_index_y:34 ,min_index_n:34
min_index_y:58 ,min_index_n:59
min_index_y:0 ,min_index_n:0
min_index_y:65 ,min_index_n:66
min_index_y:0 ,min_index_n:0
min_index_y:41 ,min_index_n:41
min_index_y:0 ,min_index_n:0
min_index_y:37 ,min_index_n:37
min_index_y:0 ,min_index_n:0
min_index_y:0 ,min_index_n:0
min_index_y:60 ,min_index_n:60
min_index_y:0 ,min_index_n:0
min_index_y:2 ,min_index_n:3
min_index_y:70 ,min_index_n:71
min_index_y:60 ,min_index_n:60
min_index_y:0 ,min_index_n:0
min_index_y:99 ,min_index_n:99
min_index_y:99 ,min_index_n:99
min_index_y:60 ,min_index_n:61
```

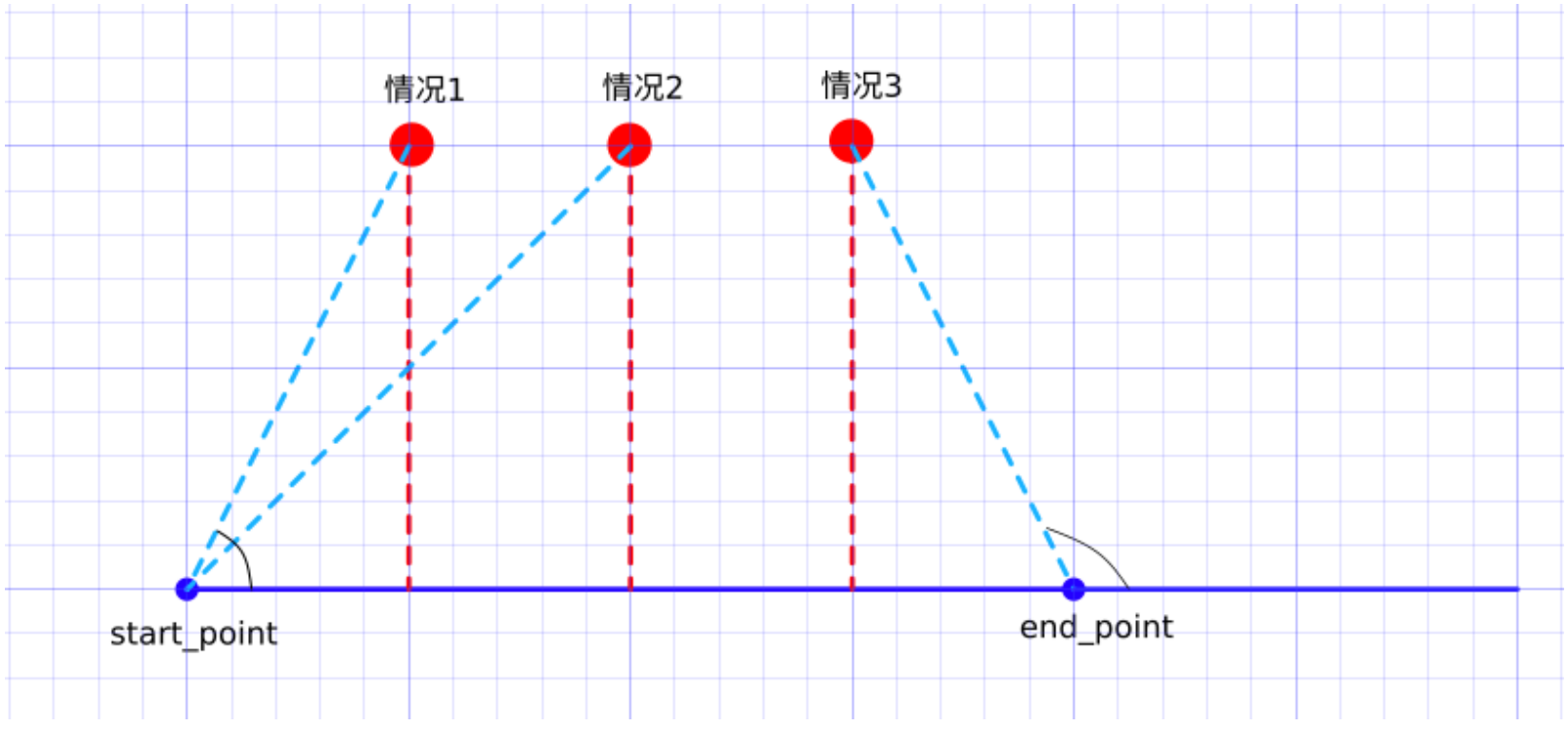
求点到线段的最小距离



$$\overrightarrow{AC} = \frac{(\overrightarrow{AP}\cdot\overrightarrow{AB})}{|\overrightarrow{AB}|^2}\ \overrightarrow{AB} = \frac{(\overrightarrow{AP}\cdot\overrightarrow{AB})}{|\overrightarrow{AB}|}\cdot\frac{\overrightarrow{AB}}{|\overrightarrow{AB}|}$$

```cpp
double LineSegment2d::DistanceSquareTo(const Vec2d &point) const {
  if (length_ <= kMathEpsilon) {
    return point.DistanceSquareTo(start_);
  }
  const double x0 = point.x() - start_.x();
  const double y0 = point.y() - start_.y();
  const double proj = x0 * unit_direction_.x() + y0 * unit_direction_.y();
  if (proj <= 0.0) {
    return Square(x0) + Square(y0);
  }
  if (proj >= length_) {
    return point.DistanceSquareTo(end_);
  }
  return Square(x0 * unit_direction_.y() - y0 * unit_direction_.x());
}
```

原因


情况1
情况2
情况3
start_point
end_point