西南交通大學
Southwest Jiaotong University

地球科学与环境工程学院

《地理信息系统原理》

课间实验报告书

设计题目：基于四叉树的轨迹数据管理系统

组　　长：于沛豪 2023113911

组　　员：姜成浩　2023114559　吴明轩 2023114306

欧阳文杰 2023114280　田翔　2023114378

朱麟飞　2023114411　万霁赋 2023114380

日　　期：2025.5.5

目　录

第一部分 实验分析

1.1 实验目的

（1）利用本节课讲授的传统四叉树方法，存储轨迹数据并建立索引

（2）研究不同树深（叶子节点存储阈值）对索引效率的影响

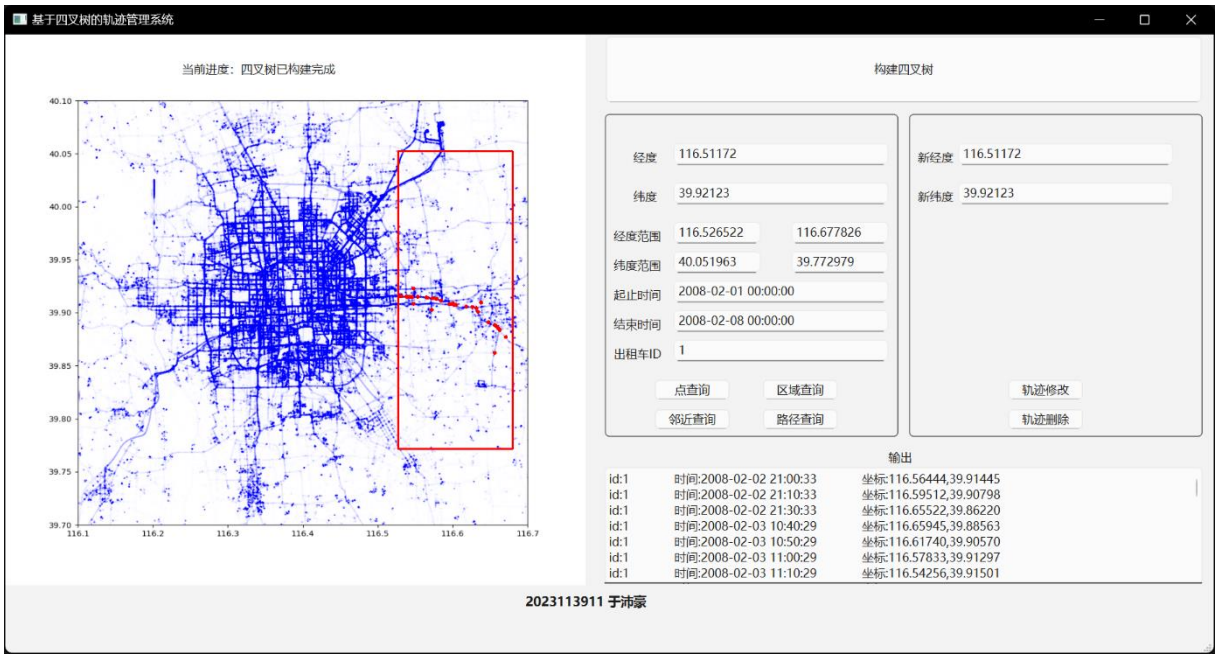
（3）分别实现点查询，区域查询（矩形区域），邻近查询，轨迹查询

（4）实现轨迹的修改，删除操作

（5）可视化所有点数据并实现交互式查询

1.1 操作环境

- 编译环境：MiniGW QT 6.8.3
- 可视化窗口展示：QT Creator



第二部分 实验流程

2.1 实验内容

- 分别实现点查询，区域查询（矩形区域），邻近查询，轨迹查询
- 实现轨迹的修改，删除操作
- 可视化所有点数据并实现交互式查询

2.2小组分工

- 程序编写：于沛豪（完成四叉树构建及基本功能） 姜成浩（完成可视化）
- 程序调试：欧阳文杰（可视化代码调试） 田翔（四叉树架构调试）
  朱麟飞（IO 读写调试）
- 实验报告编写：万霁赋（整体实验报告编写）

其中，不同成员代码之间的互相传输等通信采取 Git 进行版本管理。

以于沛豪为例，其部分 commit 记录如下所示：

```
空白@LAPTOP-S8M3UNR9 MINGW64 /e/Code/GIS/WidgetQuadtree (master)
$ git reflog
87d6aed (HEAD -> master, origin/master) HEAD@{0}: pull: Fast-forward
3291b4c HEAD@{1}: commit: 哈哈哈要做完了
aa1b184 HEAD@{2}: commit: delete build
d9a0ad1 HEAD@{3}: commit: no build
3d3bba3 HEAD@{4}: commit: readme
aaacded HEAD@{5}: pull: Merge made by the 'ort' strategy.
da7ea73 HEAD@{6}: commit: 还差修改与删除
e955482 HEAD@{7}: commit: first
7bac19c HEAD@{8}: commit: 矩形绘制
b537d5f HEAD@{9}: commit (initial): first
```

2.3 可视化处理与交互式查询

本次实验采用 QTCreator 创建 Qmake 项目， 通过槽函数以及信号来实现按钮单击事件读写数据（类似于 C#语言中的事件与委托）。

窗体布局设计如下所示：

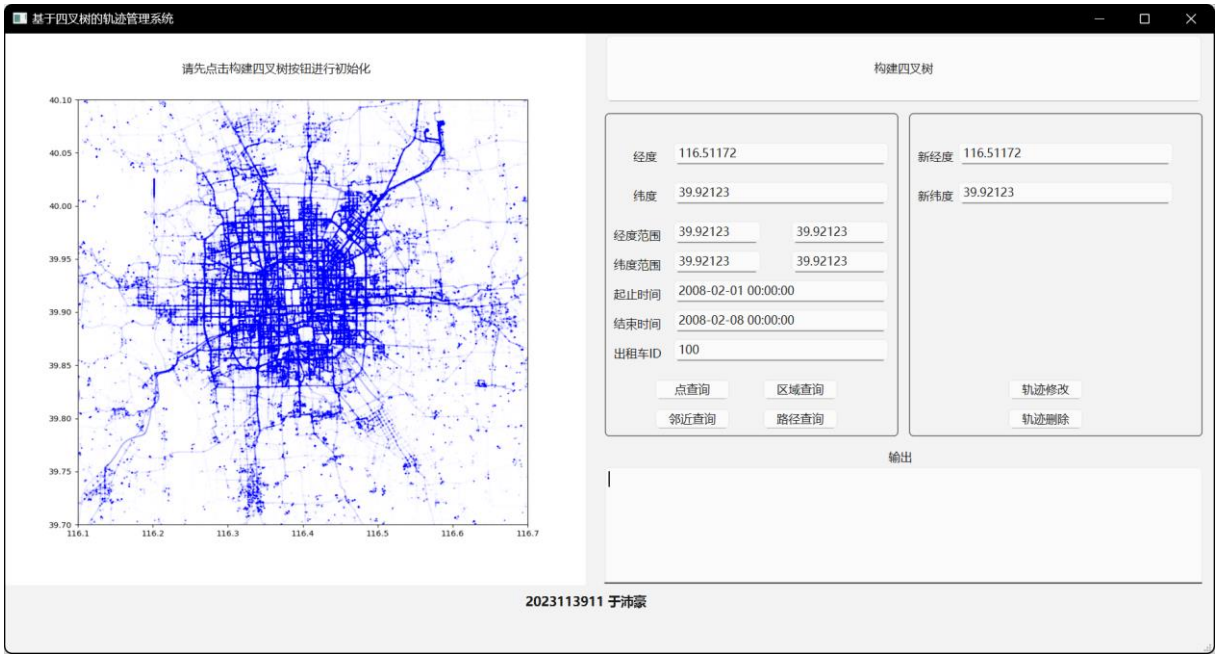


图 1　窗体布局参考

本次实验选择 Qtcretor 的 Widget 项目进行开发，其优点是适合该种小型项目，并且不宜在代码编译时窗口发生意料之外的形变。

通过窗口创建的对象如下所示：

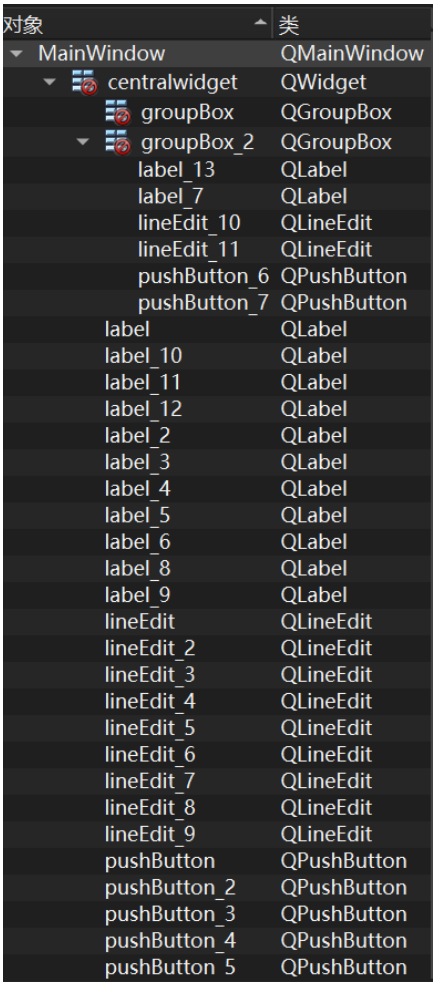


图 2　窗体对象与类

可以发现，这里轨迹图像是“预制”的，原因在于该程序采用 C 语言开发，其绘图库，即使是 Qtcreator 的绘图库也过于老旧，在加载如此数量的绘图时会直接崩溃

因此，在可视化点的时候，采用 python 代码预先生成图像，其可视化主要在于可画出出查询点的数据、查询轨迹的数据，通过点击方便实现矩形的绘制并输出经纬度到文本框中。

2.4.1 矩形显示框

当第一次单击图片时，系统将会存储点的坐标，并从窗口坐标转化为经纬度坐标。

当第二次单击图片时，系统将会以两次点击地作为对角定点，完成矩形的绘制，同时两点经纬度坐标将会被存入右侧的经度范围选框。

当第三次点击图片时，将会重置点，重新作为第一次单击图片时操作。

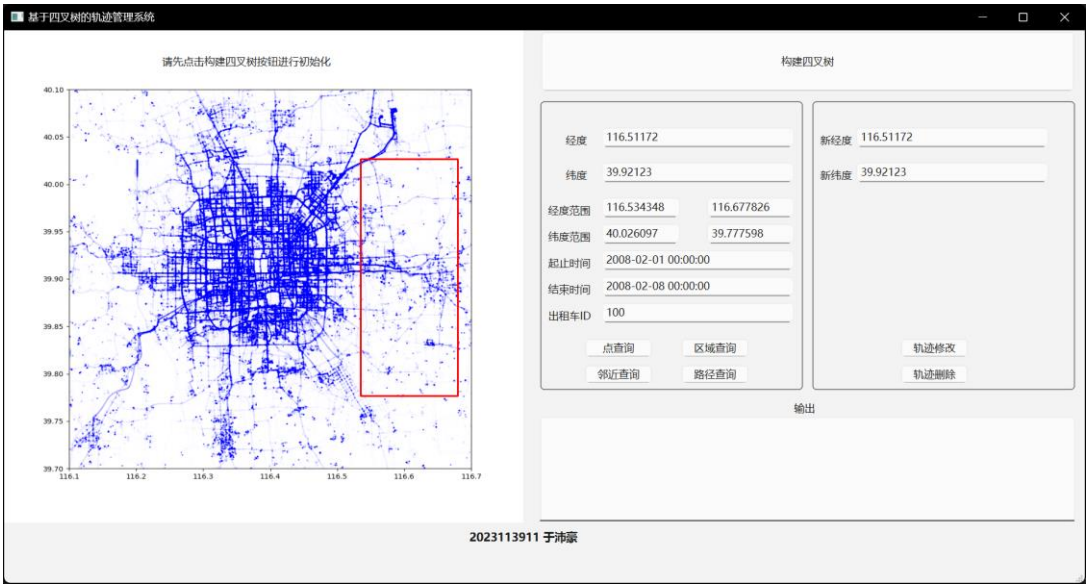


图 3　矩形选框

此时，经度范围及纬度范围即矩形选框坐标转换为经纬度值。
其绘图类代码如下所示：

```
#include "overlaywidget.h"
#include <QPainter>
#include <QMouseEvent>

OverlayWidget::OverlayWidget(QWidget *parent) : QWidget(parent) {
   // setAttribute(Qt::WA_TransparentForMouseEvents, true); // 接收鼠标事件
    setAttribute(Qt::WA_NoSystemBackground, true);
    setAttribute(Qt::WA_TranslucentBackground, true);
    setAttribute(Qt::WA_TransparentForMouseEvents, false); // 要接收鼠标事件
}

void OverlayWidget::clearPoints() {
    hasFirst = false;
    hasSecond = false;
    update();
}

void OverlayWidget::setPoint(const QPoint &pt) {
    if (!hasFirst) {
        point1 = pt;
        hasFirst = true;
        hasSecond = false;
    } else {
        point2 = pt;
        hasSecond = true;
        emit rectangleReady(point1, point2);
```

```
    }
    update();
}

void OverlayWidget::mousePressEvent(QMouseEvent *event) {
    if (hasFirst && hasSecond) {
        // 第三次点击，重置
        clearPoints();
    }
    setPoint(event->pos());
}

void OverlayWidget::paintEvent(QPaintEvent *) {
    if (hasFirst && hasSecond) {
        QPainter painter(this);
        painter.setPen(QPen(Qt::red, 2));
        QRect rect(QPoint(std::min(point1.x(), point2.x()), std::min(point1.y(
), point2.y())),
                   QPoint(std::max(point1.x(), point2.x()), std::max(point1.y(
), point2.y())));
        painter.drawRect(rect);
    }
}
```

2.4.2 点的可视化

当我们运行点查询或者路径查询时，系统将输出结果的点，并将其在图中标注出来，如下所示：

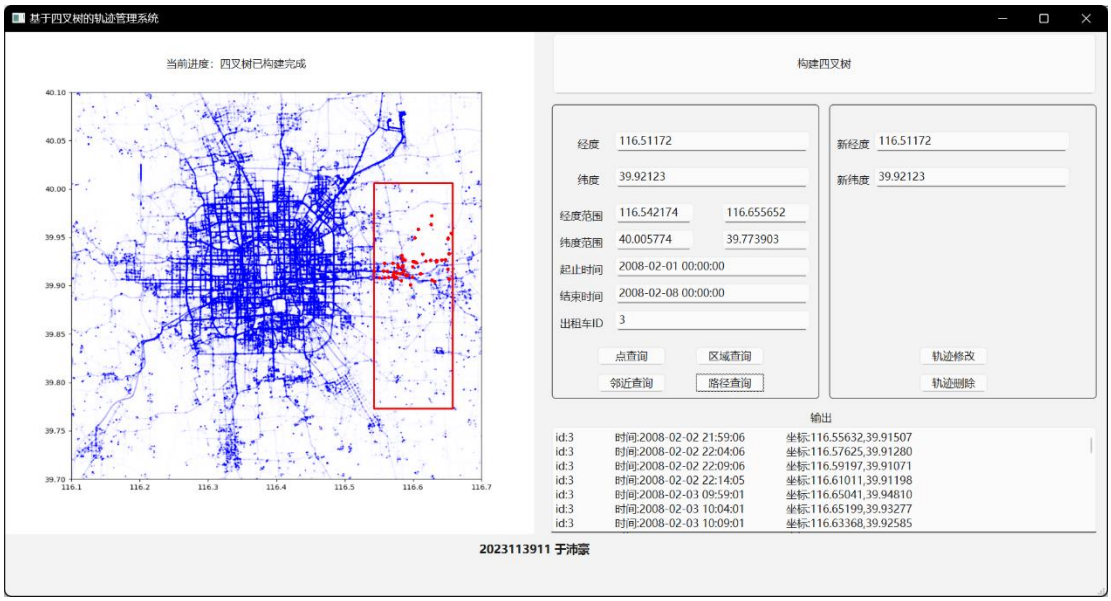


图 4　可视化点

其绘图类代码如下所示：

```
#include "drawwidget.h"
#include <QPainter>
#include <QFile>
#include <QTextStream>
#include <QDebug>

DrawWidget::DrawWidget(QWidget *parent) : QWidget(parent) {
    // 确保透明背景，避免覆盖整个控件区域
    setAttribute(Qt::WA_NoSystemBackground, true);
    setAttribute(Qt::WA_TranslucentBackground, true);
    // 确保接收鼠标事件
    setAttribute(Qt::WA_TransparentForMouseEvents, false);
}


void DrawWidget::loadPointsFromFile(const QString &filename) {
    QFile file(filename);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Failed to open file";
        return;
    }

    QTextStream in(&file);
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList parts = line.split(" 坐标:");
        if (parts.size() == 2) {
            QStringList coords = parts[1].split(",");
            if (coords.size() == 2) {
                bool ok1, ok2;
                double lon = coords[0].toDouble(&ok1);
                double lat = coords[1].toDouble(&ok2);
                if (ok1 && ok2) {
                    // 进行坐标转换
                    double windowX = (lon - 116.0047826086956) / 0.00130434782
61;
                    double windowY = (lat - 40.16189376443418) / -
0.00092378752886;
                    QPointF point(windowX, windowY);
                    points.append(point);

                    // 打印转换后的坐标，检查是否合理
                    qDebug() << "Converted point: (" << windowX << ", " << win
dowY << ")";
```

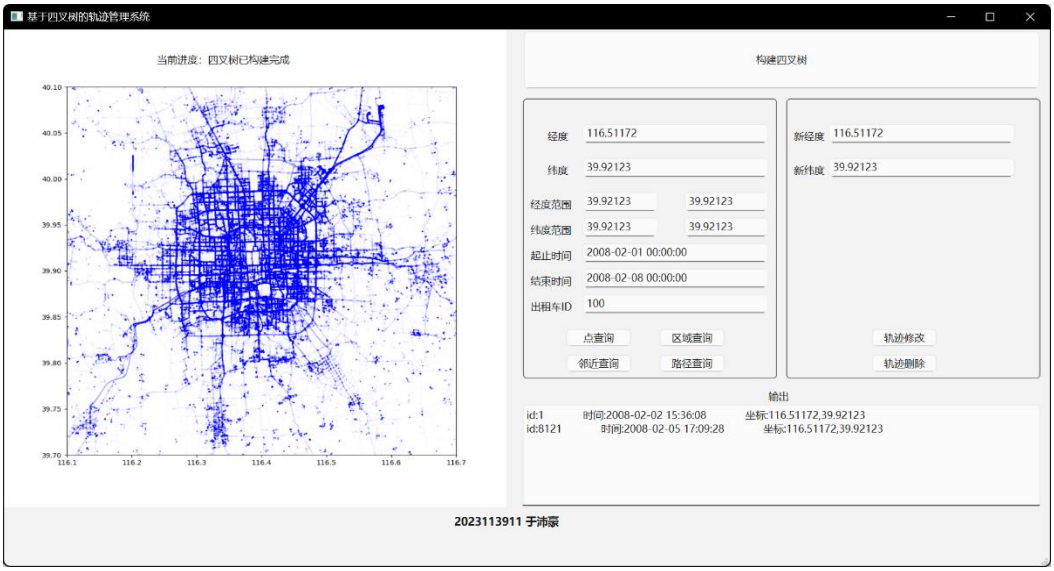
```
                }
            }
        }
    }
}

void DrawWidget::paintEvent(QPaintEvent *) {
    QPainter painter(this);
    painter.setPen(Qt::black);

    // 如果没有点要绘制，返回
    if (points.isEmpty()) {
        qDebug() << "No points to draw!";
        return;
    }

    painter.setBrush(QBrush(Qt::red));
    painter.setPen(Qt::NoPen);

    for (const QPointF &pt : points) {
        // 在每个点的位置画一个半径为 2 的红色实心圆
        painter.drawEllipse(pt, 2.0, 2.0);
    }
}
```

2.4 功能演示

2.4.1 点查询

在单击构建四叉树后。程序进行构建四叉树工作，构建完成后，图片上方显示文字改变为“四叉树已构建完成”。

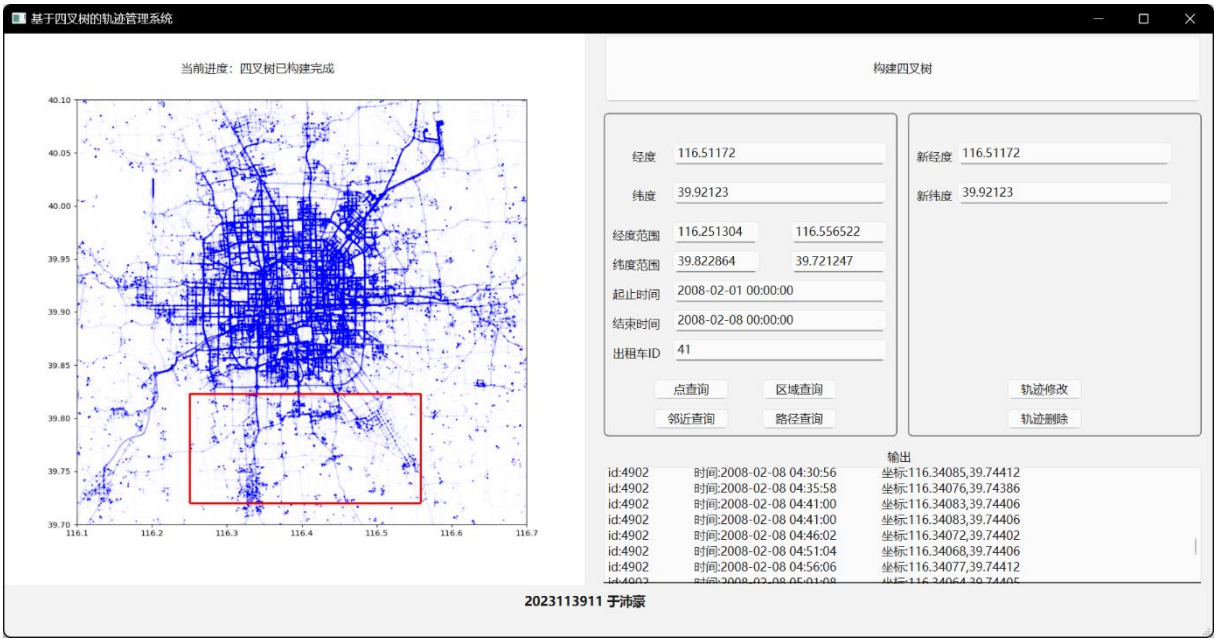
使用点查询功能，输入经纬度、输出相关行，如下所示：



2.4.2 区域查询

在图片中单击图片进行可视化，点击构建矩形的左上点与右下点，自动绘制矩形选框，并且输入矩形范围到文本框中。

此时单击区域查询，输出区域内所有数据。

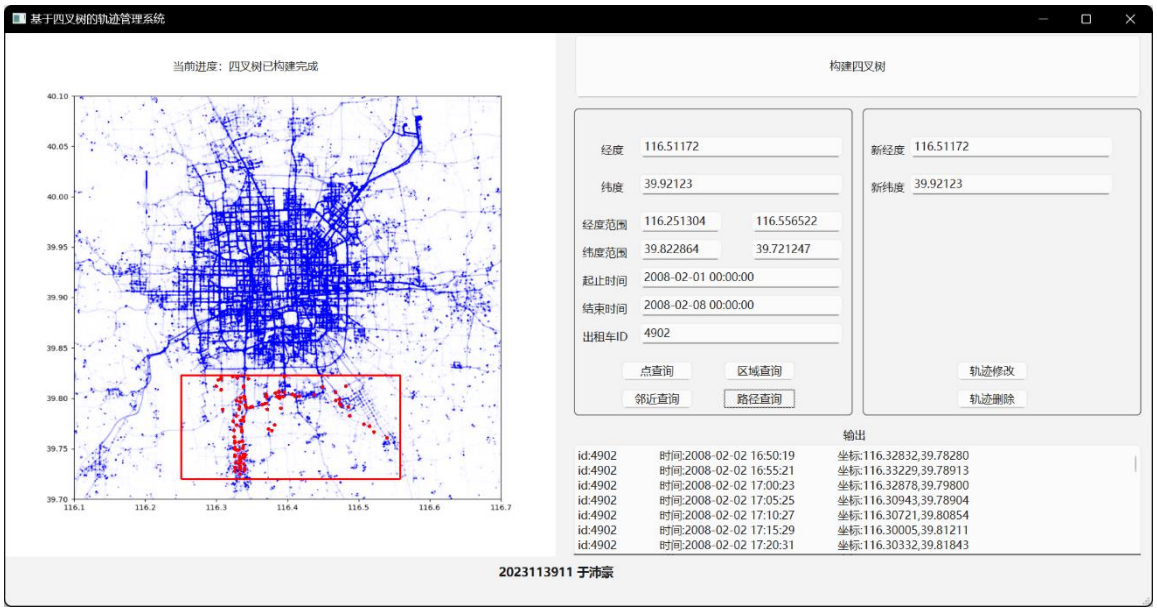


2.4.3 轨迹查询

我们可以利用的区域查询的结果，直接输入出租车 id，即可在输出框中显示轨迹，并且在图中可视化。

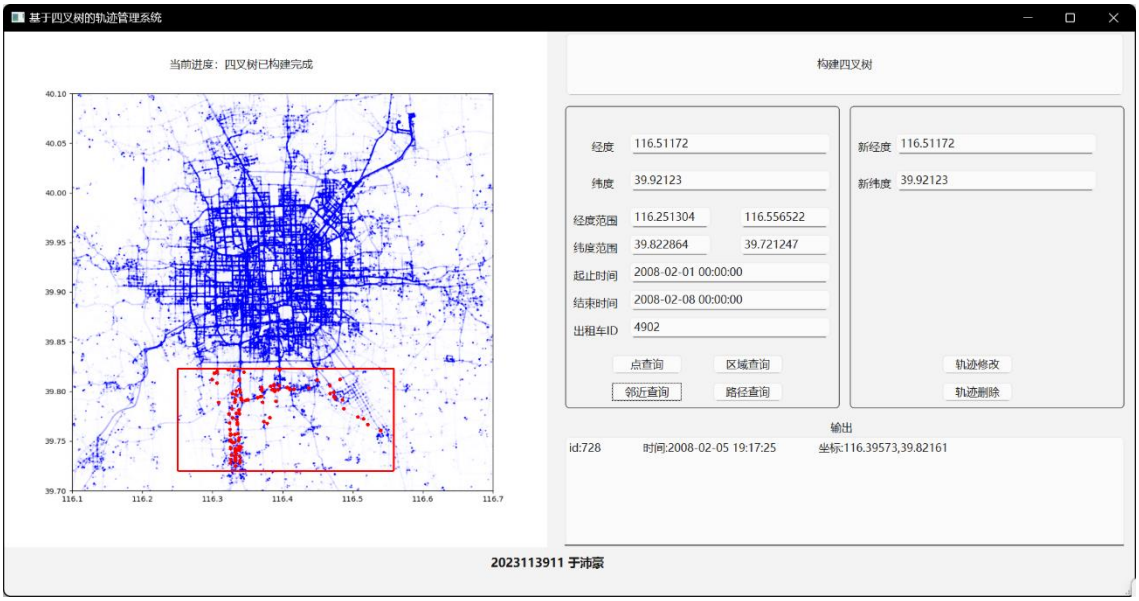
我们还可以重新建立选框，或从键盘键入经度纬度范围，输入出租车 id，输出规划及可视化，如下所示：

例如，我们搜索区域内，ID 为 4902 的出租车轨迹，结果如下：



2.4.4 邻近查询

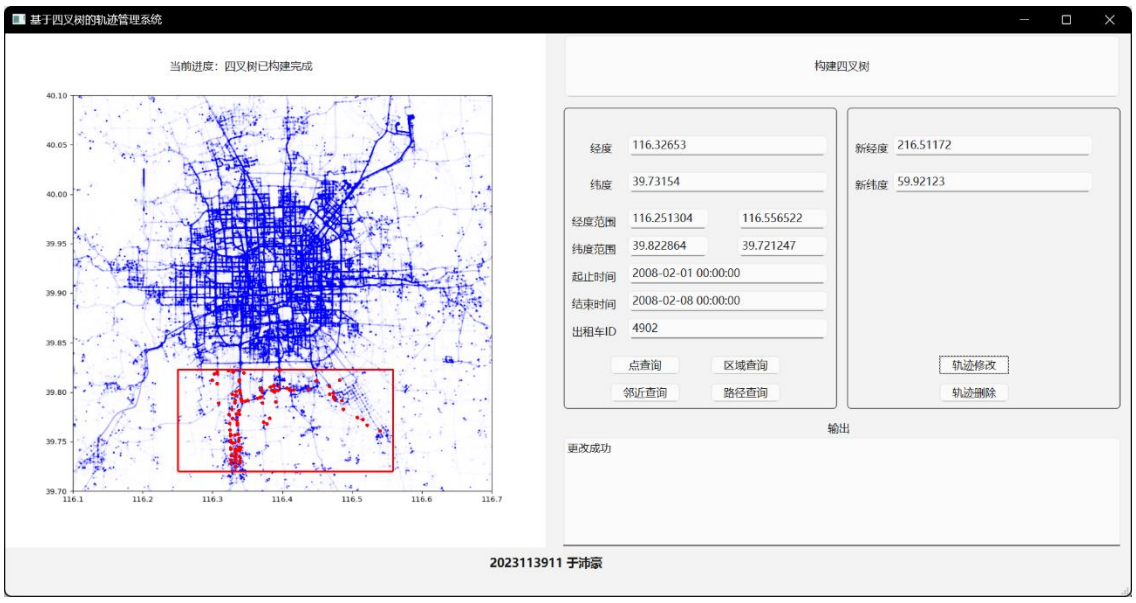
例如，我们搜索与 id 为 4902 的出租车的邻近查询，输出最近 id 出租车，如下所示：



可知，id 最近的出租车是 728，数据如输出所示。

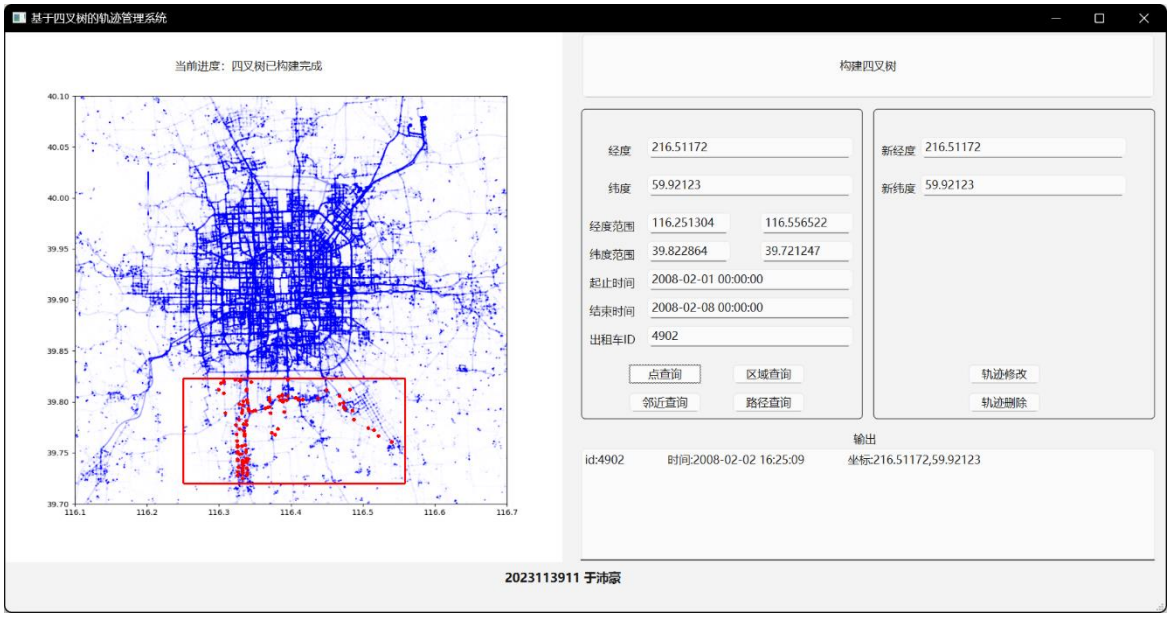
2.4.5 轨迹修改

我们可以通过前面几种方式获取到数据，而后可以在右侧输入新经度与纬度，点击轨迹修改，系统则会自动执行搜索与修改的功能，如下所示：



此时，我们重新查询轨迹修改后的结果，可以搜索的到，则可以说明，轨迹修改成功。

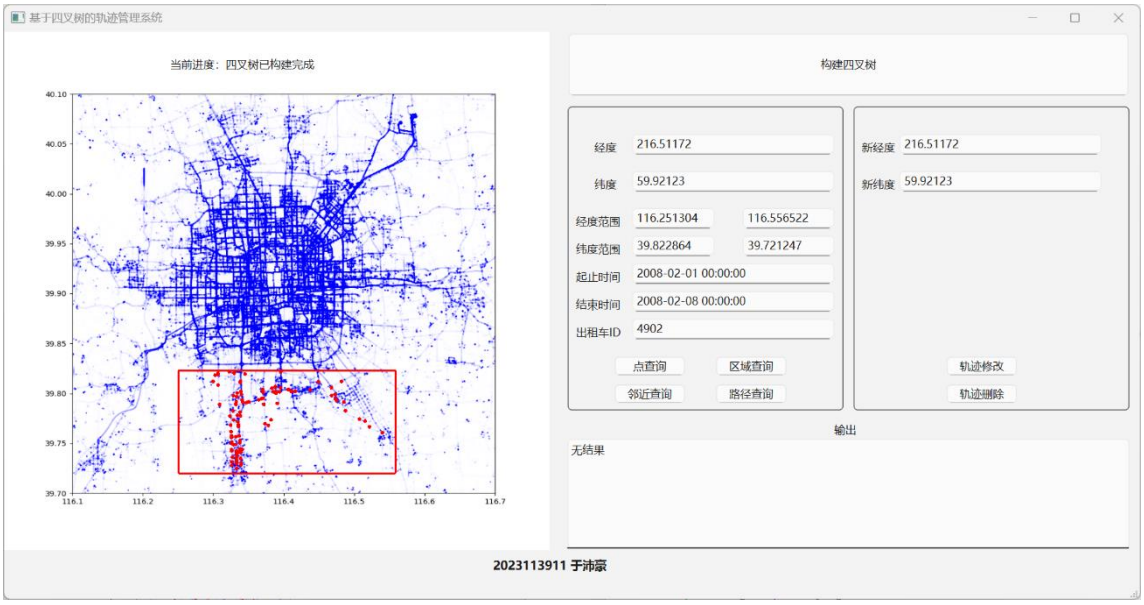
如下所示：



2.4.6 轨迹删除

同上，我们删除该轨迹。

再次搜索，可见已经搜不到结果了，说明轨迹删除成功：



由于具体实现过于复杂，将具体四叉树代码放于附录

第三部分 实验总结

3.1 关于可视化与交互式查询

本次实验采用 QTCreator 创建 Qmake 项目， 通过槽函数以及信号来实现按钮单击事件读写数据（类似于 C#语言中的事件与委托）。选择 Qtcretor 的 Widget 项目进行开发，其优点是适合该种小型项目，并且不宜在代码编译时窗口发生意料之外的形变。

在可视化点的时候，采用 python 代码预先生成图像，其可视化主要在于可画出出查询点的数据、查询轨迹的数据，通过点击方便实现矩形的绘制并输出经纬度到文本框中。

具体包括：矩形选框的交互、轨迹点绘制的交互

3.2 关于四叉树结构

关于四叉树的结构（QuadNode 类）是我们调整了最长时间的地方，其中最难的函数毫无疑问是 InsertNode，我们通过判断与根节点范围的交集（例如左上、左下、右上、右下），以此插入结点。还有就是 PointSearch 以及 AreaSearch 等函数，需要用到容器的 emplace_back 等方法，通过遍历各级四叉树的相对位置（上下左右）来确定 Search 后结果

在需要顺序读写的所有部分，我们请教了计算机专业的同学、采用容器（vector）进行存储，使得内存可以得到动态管理，加速了读写速率。

另外一个难点是 Rectangle 类，需要设计 Inside（判断点在形状内）等函数，我们利用了自带函数 make_pair 来使左上、左下、右上、右下的位置来组合数据，返回判断 bool 值。

对于交互式查询，其中需要进行坐标转换，将窗口坐标建立线性关系匹配经纬度，此处需要建立两个线性关系（经度与纬度均需建立）以匹配。

3.3实验小结

- 在本次实验中，我们深入了解了 C 语言可视化的整个操作流程，并利用 Git 进行版本管理，了解了目前现代化的编程流程
- 在课外，也自学了 C 语言利用 QtCreator 进行可视化的基本流程。这不仅对我课内的一些实验或作业有帮助，也为我今后编程打下了基础。我也加深了对 C 语言面向对象编程的基本流程的理解。

参考文献

[1]. 王远飞，何洪林. 空间数据分析方法[M]. 科学出版社, 2007.

[2]. 陈雪梅，韩洁琼. C 语言可视化编程环境的设计与实现[J]. 武汉理工大学学报：信息与管理工程版, 2010, 32(4): 561-564.

[3] 严蔚敏，吴伟民. 数据结构: C 语言版[M]. 清华大学出版社有限公司, 1997.

附录 1 MainWIndows.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <algorithm>
#include <fstream>
#include "common.h"
#include <QApplication>
#include <QFile>
#include <QTextStream>
#include <QMouseEvent>
#include <QPainter>
#include "drawwidget.h"
#pragma execution_character_set("utf-8")


MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , root(nullptr)
{
    ui->setupUi(this);

    // 设置 label 的事件过滤器，仅响应点击

    ui->label->installEventFilter(this);

    // 创建 drawWidget 作为 label 的子控件
    drawWidget = new DrawWidget(ui->label);
    drawWidget->setGeometry(ui->label->rect());
    drawWidget->setAttribute(Qt::WA_TransparentForMouseEvents);
    drawWidget->setAttribute(Qt::WA_TranslucentBackground);
    drawWidget->setStyleSheet("background: transparent;");
    drawWidget->show();

    // 创建 overlay 作为 label 的子控件
    overlay = new OverlayWidget(ui->label);
    overlay->setGeometry(ui->label->rect());
    overlay->setAttribute(Qt::WA_TranslucentBackground);
    overlay->setStyleSheet("background: transparent;");
    overlay->show();
```

```
    // 连接矩形完成信号
    connect(overlay, &OverlayWidget::rectangleReady, this, &MainWindow::updateLineEdits);
}


MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::showEvent(QShowEvent *event)
{
    QMainWindow::showEvent(event);


}

void MainWindow::on_label_2_linkActivated(const QString &link)
{

}
void MainWindow::resizeEvent(QResizeEvent *event) {
    QMainWindow::resizeEvent(event);
    if (overlay && ui->label) {
        overlay->setGeometry(ui->label->geometry());
    }
}


void MainWindow::updateLineEdits(const QPoint &p1, const QPoint &p2) {
    QPoint tl(std::min(p1.x(), p2.x()), std::min(p1.y(), p2.y()));
    QPoint br(std::max(p1.x(), p2.x()), std::max(p1.y(), p2.y()));

    // 线性变换参数
    constexpr double scale1 = 0.0013043478261;
    constexpr double offset1 = 116.0047826086956;

    constexpr double scale2 = -0.00092378752886;
    constexpr double offset2 = 40.16189376443418;

    // 应用变换
    double transformedX1 = tl.x() * scale1 + offset1;
    double transformedX2 = br.x() * scale1 + offset1;
    double transformedY1 = tl.y() * scale2 + offset2;
    double transformedY2 = br.y() * scale2 + offset2;

    ui->lineEdit_3->setText(QString::number(transformedX1, 'f', 6));  // 保留6位小数
    ui->lineEdit_4->setText(QString::number(transformedY1, 'f', 6));
    ui->lineEdit_5->setText(QString::number(transformedX2, 'f', 6));
    ui->lineEdit_6->setText(QString::number(transformedY2, 'f', 6));
}
```

```
bool MainWindow::eventFilter(QObject *watched, QEvent *event) {
    if (watched == ui->label && event->type() == QEvent::MouseButtonPress) {
        QMouseEvent *mouseEvent = static_cast<QMouseEvent*>(event);
        QPoint localPos = mouseEvent->pos();  // 鼠标相对于 label 的位置

        // 将事件坐标传递给 overlay
        if (overlay) {
            overlay->setPoint(localPos);
            overlay->update();  // 触发重绘
        }

        return true; // 表示事件已处理
    }
    return QMainWindow::eventFilter(watched, event);
}



void MainWindow::on_pushButton_clicked()
{
    QString lon = ui->lineEdit->text();
    double x = lon.toDouble();
    QString lan = ui->lineEdit_2->text();
    double y = lan.toDouble();

    qDebug() << "输入经度：" << lon << " 转换后：" << x;
    qDebug() << "输入纬度：" << lan << " 转换后：" << y;
    auto result = root->PointSearch(std::make_pair(x, y));
    ui->textEdit->setText("222");
    if (result.empty()) {
        //std::cout << "未查询到\n";
        ui->textEdit->setText("未查询到");
        ui->textEdit->setText(lon+lan);
    } else {

        //ui->textEdit->setText(result);
        ui->textEdit->setText("111");
        printVector(result);


        QFile file("output.txt"); // 你也可以写绝对路径

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            ui->textEdit->setText("无法打开文件！");
            return;
        }

        QTextStream in(&file);
        QString fileContent = in.readAll();  // 读取全部内容
        file.close();
```

```
        ui->textEdit->setText(fileContent);  // 设置到文本框中
    }


}


void MainWindow::on_pushButton_2_clicked()
{
    this->root = new QuadNode();   // 正确地使用成员变量
    int DATANUM = 10000;
    Rectangle* bounding_box = new Rectangle();
    double min_lon = 181, max_lon = 0, min_lat = 181, max_lat = 0;

    ui->label_2->setText("正在构建包围盒");

    for (int i = 1; i <= DATANUM; ++i) {
        std::string buffer;
        std::string path = "E:\\Code\\GIS\\Sichashu\\Quadtree\\data\\" + std::to_string(i)
+ ".txt";
        std::ifstream file(path);
        if (!file.is_open()) continue;

        while (std::getline(file, buffer)) {
            auto gps = str2data(buffer);
            min_lon = std::min(min_lon, gps->longitude);
            min_lat = std::min(min_lat, gps->latitude);
            max_lon = std::max(max_lon, gps->longitude);
            max_lat = std::max(max_lat, gps->latitude);
            delete gps;
        }

        file.close();
    }

    bounding_box->bottom_left = std::make_pair(min_lon, min_lat);
    bounding_box->top_right = std::make_pair(max_lon, max_lat);
    this->root->range = bounding_box;

    ui->label_2->setText("正在构建四叉树");

    for (int i = 1; i <= DATANUM; ++i) {
        std::string buffer;
        std::string path = "E:\\Code\\GIS\\Sichashu\\Quadtree\\data\\" + std::to_string(i)
+ ".txt";
        std::ifstream file(path);
        if (!file.is_open()) continue;

        while (std::getline(file, buffer)) {
            auto gps = str2data(buffer);
            this->root->InsertNode(gps, 8);  // 用 this->root 插入数据
        }

```

```
        file.close();
    }

    ui->label_2->setText("当前进度：四叉树已构建完成");
}



void MainWindow::on_pushButton_3_clicked()
{
    QString lon1 = ui->lineEdit_3->text();
    double x1 = lon1.toDouble();
    QString lan1 = ui->lineEdit_4->text();
    double y1 = lan1.toDouble();
    QString lon2 = ui->lineEdit_5->text();
    double x2 = lon2.toDouble();
    QString lan2 = ui->lineEdit_6->text();
    double y2 = lan2.toDouble();

    Rectangle* rect = new Rectangle(x2, y1, x1, y2);
    //Rectangle* rect = new Rectangle(y1, x2, y2, x1);
    auto result = root->AreaSearch(rect);

    ui->textEdit->setText("222");
    if (result.empty()) {
        //std::cout << "未查询到\n";
        ui->textEdit->setText("未查询到");
       // ui->textEdit->setText(lon+lan);
    } else {

        //ui->textEdit->setText(result);
        ui->textEdit->setText("111");
        printVector(result);


        QFile file("output.txt"); // 你也可以写绝对路径

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            ui->textEdit->setText("无法打开文件！");
            return;
        }

        QTextStream in(&file);
        QString fileContent = in.readAll();  // 读取全部内容
        file.close();

        ui->textEdit->setText(fileContent);  // 设置到文本框中
    }
}


void MainWindow::on_pushButton_4_clicked()
{
```

```
    QString lon1 = ui->lineEdit_3->text();
    double x1 = lon1.toDouble();
    QString lan1 = ui->lineEdit_4->text();
    double y1 = lan1.toDouble();
    QString lon2 = ui->lineEdit_5->text();
    double x2 = lon2.toDouble();
    QString lan2 = ui->lineEdit_6->text();
    double y2 = lan2.toDouble();

    Rectangle* rect = new Rectangle(x2, y1, x1, y2);

    std::vector<int> time1, time2;

    QString timeStr1 = ui->lineEdit_7->text();  // 例如 "2025-05-05 14:30:15"
    QStringList parts1 = timeStr1.split(" ");
    if (parts1.size() == 2) {
        QStringList date1 = parts1[0].split("-");
        QStringList clock1 = parts1[1].split(":");
        if (date1.size() == 3 && clock1.size() == 3) {
            time1.push_back(date1[0].toInt()); // 年
            time1.push_back(date1[1].toInt()); // 月
            time1.push_back(date1[2].toInt()); // 日
            time1.push_back(clock1[0].toInt()); // 时
            time1.push_back(clock1[1].toInt()); // 分
            time1.push_back(clock1[2].toInt()); // 秒
        }
    }

    QString timeStr2 = ui->lineEdit_8->text();  // 例如 "2025-05-05 15:01:30"
    QStringList parts2 = timeStr2.split(" ");
    if (parts2.size() == 2) {
        QStringList date2 = parts2[0].split("-");
        QStringList clock2 = parts2[1].split(":");
        if (date2.size() == 3 && clock2.size() == 3) {
            time2.push_back(date2[0].toInt()); // 年
            time2.push_back(date2[1].toInt()); // 月
            time2.push_back(date2[2].toInt()); // 日
            time2.push_back(clock2[0].toInt()); // 时
            time2.push_back(clock2[1].toInt()); // 分
            time2.push_back(clock2[2].toInt()); // 秒
        }
    }
    QString idxString = ui->lineEdit_9->text();
    double idx = idxString.toDouble();

    auto result = root->AdjacentSearch(rect, time1, time2, idx);

    ui->textEdit->setText("222");
    if (result == nullptr) {
        //std::cout << "未查询到\n";
        ui->textEdit->setText("未查询到");
        // ui->textEdit->setText(lon+lan);
    } else {
```

```
        //ui->textEdit->setText(result);
        ui->textEdit->setText("111");
        //printVector(result);
        QFile::remove("output.txt");
        result->print();


        QFile file("output.txt");

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            ui->textEdit->setText("无法打开文件！");
            return;
        }

        QTextStream in(&file);
        QString fileContent = in.readAll();  // 读取全部内容
        file.close();

        ui->textEdit->setText(fileContent);  // 设置到文本框中
    }
}



void MainWindow::on_pushButton_5_clicked()
{
    QString lon1 = ui->lineEdit_3->text();
    double x1 = lon1.toDouble();
    QString lan1 = ui->lineEdit_4->text();
    double y1 = lan1.toDouble();
    QString lon2 = ui->lineEdit_5->text();
    double x2 = lon2.toDouble();
    QString lan2 = ui->lineEdit_6->text();
    double y2 = lan2.toDouble();
    QString idxString = ui->lineEdit_9->text();
    int idx = idxString.toInt();
    Rectangle* rect = new Rectangle(x2, y1, x1, y2);
    //Rectangle* rect = new Rectangle(y1, x2, y2, x1);
    auto result = root->TrajectorySearch(rect,idx);

    ui->textEdit->setText("222");
    if (result.empty()) {
        ui->textEdit->setText("未查询到");
        // ui->textEdit->setText(lon+lan);
    } else {

        //ui->textEdit->setText(result);
        ui->textEdit->setText("111");
        printVector(result);


        QFile file("output.txt");
```

```
        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            ui->textEdit->setText("无法打开文件！");
            return;
        }

        QTextStream in(&file);
        QString fileContent = in.readAll();  // 读取全部内容
        file.close();

        ui->textEdit->setText(fileContent);  // 设置到文本框中

        drawWidget->setAttribute(Qt::WA_TransparentForMouseEvents, false); // 确保不拦截鼠标事件


        drawWidget->loadPointsFromFile("output.txt");
        drawWidget->update();  // 触发重绘

    }
}


void MainWindow::on_pushButton_6_clicked()
{
    QString lon = ui->lineEdit->text();
    double x = lon.toDouble();
    QString lan = ui->lineEdit_2->text();
    double y = lan.toDouble();
    QString newlon = ui->lineEdit_11->text();
    double newx = newlon.toDouble();
    QString newlan = ui->lineEdit_10->text();
    double newy = newlan.toDouble();
    QString idx = ui->lineEdit_9->text();
    int id = idx.toDouble();
    qDebug() << "输入经度：" << lon << " 转换后：" << x;
    qDebug() << "输入纬度：" << lan << " 转换后：" << y;
    auto result = root->PointChange(id,std::make_pair(x, y),newx,newy);
    ui->textEdit->setText("222");
    if (result == false) {
        ui->textEdit->setText("未查询到");
    } else {
        ui->textEdit->setText("更改成功");

    }
}


void MainWindow::on_pushButton_7_clicked()
{
    QString lon = ui->lineEdit->text();
    double x = lon.toDouble();
    QString lan = ui->lineEdit_2->text();
```

```
    double y = lan.toDouble();
    QString idx = ui->lineEdit_9->text();
    int id = idx.toDouble();
    qDebug() << "输入经度：" << lon << " 转换后：" << x;
    qDebug() << "输入纬度：" << lan << " 转换后：" << y;
    auto result = root->PointDelete(id,x, y);
    ui->textEdit->setText("222");
    if (result == false) {
        //std::cout << "未查询到\n";
        ui->textEdit->setText("未查询到");
        // ui->textEdit->setText(lon+lan);
    } else {

        //ui->textEdit->setText(result);
        ui->textEdit->setText("删除成功");

    }
}
```

附录 2 quadtree.cpp

```
#include "quadtree.h"
#include <float.h>
#include <cmath>
#include <iomanip>
#include <iostream>
#include "common.h"
#include <fstream>
GPSdata::GPSdata() {}

GPSdata::GPSdata(int _id, string _time, double _lon, double _lat)
    : id(_id), time(_time), longitude(_lon), latitude(_lat) {}

void GPSdata::print() {
    std::ofstream outFile("output.txt", std::ios::app); // 追加写入

    if (!outFile.is_open()) {
        std::cerr << "无法打开输出文件\n";
        return;
    }

    outFile << "id:" << this->id <<std::setw(20)<< "时间:" << this->time<<std::setw(20)<<"坐
标:" << std::fixed << std::setprecision(5)
            << this->longitude << ',' << std::fixed << std::setprecision(5)
            << this->latitude << '\n';
    //outFile << "-------------------------\n";

    outFile.close();
}
```

```
Rectangle::Rectangle() {}

Rectangle::Rectangle(point2d _top_right, point2d _bottom_left)
    : top_right(_top_right), bottom_left(_bottom_left) {}

Rectangle::Rectangle(double top_right_x,
                     double top_right_y,
                     double bottom_left_x,
                     double bottom_left_y) {
    top_right = std::make_pair(top_right_x, top_right_y);
    bottom_left = std::make_pair(bottom_left_x, bottom_left_y);
}

Rectangle::Rectangle(point2d _top_right,
                     double bottom_left_x,
                     double bottom_left_y)
    : top_right(_top_right) {
    bottom_left = std::make_pair(bottom_left_x, bottom_left_y);
}
Rectangle::Rectangle(double top_right_x,
                     double top_right_y,
                     point2d _bottom_left)
    : bottom_left(_bottom_left) {
    top_right = std::make_pair(top_right_x, top_right_y);
}

bool Rectangle::Inside(point2d p) {
    auto x = p.first;
    auto y = p.second;
    return x >= bottom_left.first && x <= top_right.first &&
           y <= top_right.second && y >= bottom_left.second;
}

bool Rectangle::Inside(double x, double y) {
    return (x >= bottom_left.first || equal(x, bottom_left.first)) &&
           (x <= top_right.first || equal(x, top_right.first)) &&
           (y <= top_right.second || equal(y, top_right.second)) &&
           (y >= bottom_left.second || equal(y, bottom_left.second));
}

QuadNode::QuadNode() {}

QuadNode::QuadNode(Rectangle* bounding) : range(bounding) {}

void QuadNode::InsertNode(GPSdata* gpsdata, int depth) {
    // 空节点不用插入
    if (gpsdata == nullptr) {
        return;
    }

    double node_x = gpsdata->longitude;
    double node_y = gpsdata->latitude;
```

```
    // 要插入的节点不在范围内
    if (!this->range->Inside(node_x, node_y)) {
        return;
    }

    double root_x1 = this->range->top_right.first;
    double root_y1 = this->range->top_right.second;
    double root_x2 = this->range->bottom_left.first;
    double root_y2 = this->range->bottom_left.second;
    double x_center = (root_x1 + root_x2) / 2;
    double y_center = (root_y1 + root_y2) / 2;

    if (depth == 0) {
        this->data.push_back(gpsdata);
        return;
    }

    // 在根节点范围的左半部分
    if (node_x <= x_center) {
        // 在左上角
        if (node_y >= y_center) {
            if (this->child[LEFTTOP] == nullptr) {
                Rectangle* rect =
                    new Rectangle(x_center, root_y1, root_x2, y_center);
                this->child[LEFTTOP] = new QuadNode(rect);
                this->child[LEFTTOP]->father = this;
            }
            this->child[LEFTTOP]->InsertNode(gpsdata, depth - 1);
        }
        // 在左下角
        else {
            if (this->child[LEFTBOT] == nullptr) {
                Rectangle* rect =
                    new Rectangle(x_center, y_center, root_x2, root_y2);
                this->child[LEFTBOT] = new QuadNode(rect);
                this->child[LEFTBOT]->father = this;
            }
            this->child[LEFTBOT]->InsertNode(gpsdata, depth - 1);
        }
    }
    // 在根节点范围的右半部分
    else {
        // 在右上角
        if (node_y >= y_center) {
            if (this->child[RIGHTTOP] == nullptr) {
                Rectangle* rect =
                    new Rectangle(root_x1, root_y1, x_center, y_center);
                this->child[RIGHTTOP] = new QuadNode(rect);
                this->child[RIGHTTOP]->father = this;
            }
            this->child[RIGHTTOP]->InsertNode(gpsdata, depth - 1);
        }
        // 在右下角
```

```
        else {
            if (this->child[RIGHTBOT] == nullptr) {
                Rectangle* rect =
                    new Rectangle(root_x1, y_center, x_center, root_y2);
                this->child[RIGHTBOT] = new QuadNode(rect);
                this->child[RIGHTBOT]->father = this;
            }
            this->child[RIGHTBOT]->InsertNode(gpsdata, depth - 1);
        }
    }
}

QuadNode* QuadNode::Search(point2d p) {
    return nullptr;
}

bool QuadNode::isLeaf() {
    return child[0] == nullptr && child[1] == nullptr && child[2] == nullptr &&
           child[3] == nullptr;
}

void QuadNode::findPointLeaf(point2d p, vector<GPSdata*>& leaf) {
    if (this->isLeaf()) {
        leaf = this->data;
    } else {
        if (this->child[LEFTTOP] != nullptr &&
            this->child[LEFTTOP]->range->Inside(p)) {
            this->child[LEFTTOP]->findPointLeaf(p, leaf);
        } else if (this->child[LEFTBOT] != nullptr &&
                   this->child[LEFTBOT]->range->Inside(p)) {
            this->child[LEFTBOT]->findPointLeaf(p, leaf);
        } else if (this->child[RIGHTTOP] != nullptr &&
                   this->child[RIGHTTOP]->range->Inside(p)) {
            this->child[RIGHTTOP]->findPointLeaf(p, leaf);
        } else if (this->child[RIGHTBOT] != nullptr &&
                   this->child[RIGHTBOT]->range->Inside(p)) {
            this->child[RIGHTBOT]->findPointLeaf(p, leaf);
        }
    }
}

vector<GPSdata*> QuadNode::PointSearch(point2d p) {
    vector<GPSdata*> leaf;
    this->findPointLeaf(p, leaf);
    vector<GPSdata*> result;
    // std::cout << leaf.size();
    //printVector(leaf, "E:/leaf.txt");
    for (auto dat : leaf) {
        if (equal(dat->longitude, p.first) && equal(dat->latitude, p.second)) {
            result.emplace_back(dat);
        }
    }
```

```
    return result;
}

vector<GPSdata*> QuadNode::AreaSearch(Rectangle* rect) {
    point2d p1 = rect->top_right;
    point2d p2 = rect->bottom_left;
    point2d p3 =
        std::make_pair(rect->top_right.first, rect->bottom_left.second);
    point2d p4 =
        std::make_pair(rect->bottom_left.first, rect->top_right.second);

    vector<vector<GPSdata*>> leafs;
    vector<GPSdata*> leaf1, leaf2, leaf3, leaf4;
    this->findPointLeaf(p1, leaf1);
    this->findPointLeaf(p2, leaf2);
    this->findPointLeaf(p3, leaf3);
    this->findPointLeaf(p4, leaf4);
    leafs.emplace_back(leaf1);
    leafs.emplace_back(leaf2);
    leafs.emplace_back(leaf3);
    leafs.emplace_back(leaf4);

    vector<GPSdata*> temp = {leafs[0].empty() ? nullptr : leafs[0][0],
                             leafs[1].empty() ? nullptr : leafs[1][0],
                             leafs[2].empty() ? nullptr : leafs[2][0],
                             leafs[3].empty() ? nullptr : leafs[3][0]};
    vector<int> same(4, 0);
    int k = 1;
    for (int i = 0; i < 4; ++i) {
        if (same[i] == 0) {
            same[i] = k;
            ++k;
        }

        for (int j = 0; j < 4; ++j) {
            // 这块是空的就设置为5，因为如果不是空的最大也就是4
            if (temp[i] == nullptr) {
                same[i] = 5;
            } else {
                if (same[j] == 0 && temp[i]->time == temp[j]->time) {
                    same[j] = same[i];
                }
            }
        }
    }

    k = 1;
    vector<GPSdata*> result;
    for (int i = 0; i < 4; ++i) {
        if (same[i] == k) {
            ++k;
            for (auto dat : leafs[i]) {
                if (rect->Inside(dat->longitude, dat->latitude)) {
```

```
                    result.emplace_back(dat);
                }
            }
        }
    }

    return result;
}

vector<GPSdata*> QuadNode::TrajectorySearch(Rectangle* rect, int target_idx) {
    point2d p1 = rect->top_right;
    point2d p2 = rect->bottom_left;
    point2d p3 =
        std::make_pair(rect->top_right.first, rect->bottom_left.second);
    point2d p4 =
        std::make_pair(rect->bottom_left.first, rect->top_right.second);

    vector<vector<GPSdata*>> leafs;
    vector<GPSdata*> leaf1, leaf2, leaf3, leaf4;
    this->findPointLeaf(p1, leaf1);
    this->findPointLeaf(p2, leaf2);
    this->findPointLeaf(p3, leaf3);
    this->findPointLeaf(p4, leaf4);
    leafs.emplace_back(leaf1);
    leafs.emplace_back(leaf2);
    leafs.emplace_back(leaf3);
    leafs.emplace_back(leaf4);

    vector<GPSdata*> temp = {leafs[0].empty() ? nullptr : leafs[0][0],
                             leafs[1].empty() ? nullptr : leafs[1][0],
                             leafs[2].empty() ? nullptr : leafs[2][0],
                             leafs[3].empty() ? nullptr : leafs[3][0]};
    vector<int> same(4, 0);
    int k = 1;
    for (int i = 0; i < 4; ++i) {
        if (same[i] == 0) {
            same[i] = k;
            ++k;
        }

        for (int j = 0; j < 4; ++j) {
            if (temp[i] == nullptr) {
                same[i] = 5;
            } else {
                if (same[j] == 0 && temp[i]->time == temp[j]->time) {
                    same[j] = same[i];
                }
            }
        }
    }

    k = 1;
    vector<GPSdata*> result;
```

```
    for (int i = 0; i < 4; ++i) {
        if (same[i] == k) {
            ++k;
            for (auto dat : leafs[i]) {
                if (dat && rect->Inside(dat->longitude, dat->latitude) && dat->id == target
_idx) {
                    result.emplace_back(dat);
                }
            }
        }
    }

    return result;
}

GPSdata* QuadNode::AdjacentSearch(Rectangle* rect,
                                  vector<int> time1,
                                  vector<int> time2,
                                  int idx) {
    vector<GPSdata*> area = AreaSearch(rect);
    vector<GPSdata*> taxi_this;
    vector<GPSdata*> taxi_other;
    GPSdata* result;
    double min_dis = DBL_MAX;
    for (auto gps : area) {
        if (later(gps->time, time1) && earlier(gps->time, time2)) {
            if (gps->id == idx) {
                taxi_this.emplace_back(gps);
            } else {
                taxi_other.emplace_back(gps);
            }
        }
    }

    for (auto other_taxi : taxi_other) {
        for (auto this_taxi : taxi_this) {
            double d = distance(other_taxi, this_taxi);

            if (d < min_dis) {
                min_dis = d;
                result = other_taxi;
            }
        }
    }
    return result;
}

bool QuadNode::PointDelete(int id, double lon, double lat) {
    // 如果当前区域不包含该点，直接返回失败
    if (!range->Inside(lon, lat)) {
        return false;
    }
```

```
    // 如果是叶子节点，直接在当前节点的 data 中查找并删除
    if (isLeaf()) {
        for (auto it = data.begin(); it != data.end(); ++it) {
            if ((*it)->id == id && equal((*it)->longitude, lon) && equal((*it)->latitude, l
at)) {
                delete *it; // 释放内存
                data.erase(it);
                return true;
            }
        }
        return false; // 未找到匹配的点
    } else {
        // 确定子节点位置
        double x_center = (range->top_right.first + range->bottom_left.first) / 2;
        double y_center = (range->top_right.second + range->bottom_left.second) / 2;

        int index = -1;
        if (lon <= x_center) {
            index = (lat >= y_center) ? LEFTTOP : LEFTBOT;
        } else {
            index = (lat >= y_center) ? RIGHTTOP : RIGHTBOT;
        }

        // 如果子节点存在，递归删除
        if (child[index] != nullptr) {
            return child[index]->PointDelete(id, lon, lat);
        } else {
            // 子节点不存在，说明数据存储在当前节点（可能未分割）
            for (auto it = data.begin(); it != data.end(); ++it) {
                if ((*it)->id == id && equal((*it)->longitude, lon) && equal((*it)->latitud
e, lat)) {
                    delete *it;
                    data.erase(it);
                    return true;
                }
            }
            return false;
        }
    }
}

bool QuadNode::PointDelete(int id, string time) {
    bool deleted = false;

    if (!isLeaf()) {
        for (int i = 0; i < 4; ++i) {
            if (child[i] != nullptr) {
                deleted = child[i]->PointDelete(id, time);
                if (deleted) return true;
            }
        }
    }
```

```
    for (auto it = data.begin(); it != data.end(); ++it) {
        if ((*it)->id == id && (*it)->time == time) {
            delete *it;
            data.erase(it);
            return true;
        }
    }

    return deleted;
}

bool QuadNode::PointChange(int id, point2d p, double new_lon, double new_lat) {
    // 旧坐标p对应的点
    GPSdata* old_data = nullptr;

    // 找到旧坐标p的叶子节点
    vector<GPSdata*> leaf_data;
    findPointLeaf(p, leaf_data);

    // 查找旧数据点
    for (auto data : leaf_data) {
        if (data->id == id && equal(data->longitude, p.first) && equal(data->latitude, p.se
cond)) {
            old_data = data;
            break;
        }
    }

    if (old_data == nullptr) {
        return false; // 未找到旧数据点
    }

    // 删除旧数据点
    if (!PointDelete(id, p.first, p.second)) {
        return false;
    }

    // 创建新数据点并插入四叉树
    GPSdata* new_data = new GPSdata(id, old_data->time, new_lon, new_lat);
   // delete old_data; // 释放旧数据内存

    // 插入新数据点（需要重新构建四叉树插入逻辑）
    // 假设根节点为this，最大深度为某个固定值（如10）
    InsertNode(new_data, 10);
    return true;
}

//bool QuadNode::PointChange(int id, point2d p, double lon, double lat) {}

//bool QuadNode::PointDelete(int id, double lon, double lat) {}

//bool QuadNode::PointDelete(int id, string time) {}
```

附录 3 overlaywidget.cpp

```cpp
#include "overlaywidget.h"
#include <QPainter>
#include <QMouseEvent>

OverlayWidget::OverlayWidget(QWidget *parent) : QWidget(parent) {
   // setAttribute(Qt::WA_TransparentForMouseEvents, true); // 接收鼠标事件
    setAttribute(Qt::WA_NoSystemBackground, true);
    setAttribute(Qt::WA_TranslucentBackground, true);
    setAttribute(Qt::WA_TransparentForMouseEvents, false); // 要接收鼠标事件
}

void OverlayWidget::clearPoints() {
    hasFirst = false;
    hasSecond = false;
    update();
}

void OverlayWidget::setPoint(const QPoint &pt) {
    if (!hasFirst) {
        point1 = pt;
        hasFirst = true;
        hasSecond = false;
    } else {
        point2 = pt;
        hasSecond = true;
        emit rectangleReady(point1, point2);
    }
    update();
}

void OverlayWidget::mousePressEvent(QMouseEvent *event) {
    if (hasFirst && hasSecond) {
        // 第三次点击，重置
        clearPoints();
    }
    setPoint(event->pos());
}

void OverlayWidget::paintEvent(QPaintEvent *) {
    if (hasFirst && hasSecond) {
        QPainter painter(this);
        painter.setPen(QPen(Qt::red, 2));
        QRect rect(QPoint(std::min(point1.x(), point2.x()), std::min(point1.y(), point2.y()
)),
                   QPoint(std::max(point1.x(), point2.x()), std::max(point1.y(), point2.y()
)));
        painter.drawRect(rect);
    }
}
```

附录 4 drawwidget.cpp

```cpp
#include "drawwidget.h"
#include <QPainter>
#include <QFile>
#include <QTextStream>
#include <QDebug>

DrawWidget::DrawWidget(QWidget *parent) : QWidget(parent) {
    // 确保透明背景，避免覆盖整个控件区域
    setAttribute(Qt::WA_NoSystemBackground, true);
    setAttribute(Qt::WA_TranslucentBackground, true);
    // 确保接收鼠标事件
    setAttribute(Qt::WA_TransparentForMouseEvents, false);
}


void DrawWidget::loadPointsFromFile(const QString &filename) {
    QFile file(filename);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Failed to open file";
        return;
    }

    QTextStream in(&file);
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList parts = line.split(" 坐标:");
        if (parts.size() == 2) {
            QStringList coords = parts[1].split(",");
            if (coords.size() == 2) {
                bool ok1, ok2;
                double lon = coords[0].toDouble(&ok1);
                double lat = coords[1].toDouble(&ok2);
                if (ok1 && ok2) {
                    // 进行坐标转换
                    double windowX = (lon - 116.0047826086956) / 0.00130434782
61;
                    double windowY = (lat - 40.16189376443418) / -
0.00092378752886;
                    QPointF point(windowX, windowY);
                    points.append(point);

                    // 打印转换后的坐标，检查是否合理
                    qDebug() << "Converted point: (" << windowX << ", " << win
dowY << ")";
                }
            }
        }
    }
}
```

```cpp
void DrawWidget::paintEvent(QPaintEvent *) {
    QPainter painter(this);
    painter.setPen(Qt::black);

    // 如果没有点要绘制，返回
    if (points.isEmpty()) {
        qDebug() << "No points to draw!";
        return;
    }

    painter.setBrush(QBrush(Qt::red));
    painter.setPen(Qt::NoPen);

    for (const QPointF &pt : points) {
        // 在每个点的位置画一个半径为 2 的红色实心圆
        painter.drawEllipse(pt, 2.0, 2.0);
    }
}
```