Vector Data Compression Algorithms

# 矢量数据压缩算法

INFORMATION ENGINEERING UNIVERSITY

# 中国人民解放军战略支援部队信息工程大学—李响副教授

PLA Strategic Support Force Information Engineering University——A/Prof. Xiang Li

- 德国奥格斯堡大学访问学者和青年科学家，地理信息世界特聘审稿专家，测绘学报等核心期刊审稿人，高校GIS论坛十大新锐人物。
- 主要研究方向地理信息系统平台及其应用，主持国家自然科学基金，国家重点研发（子课题）等课题多项，获军事理论成果一等奖1项，军队科技进步二等奖2项，三等奖1项，高校GIS论坛“优秀教学成果”奖1项。
- 出版和翻译著作6部，近5年，以第一作者或通讯作者发表论文16篇，发明专利2项，软件著作权3项。

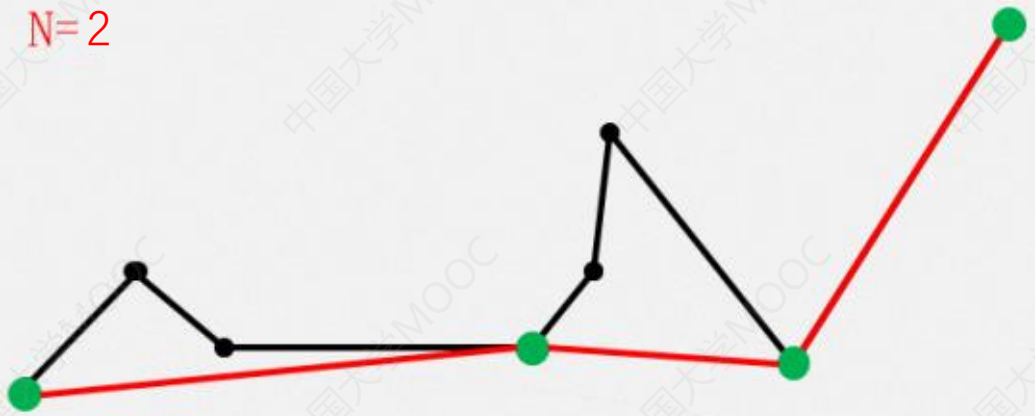
# 矢量数据压缩算法

Data Compression Algorithms

## 间隔取点法（Nth Point）

**算 法 思 路**

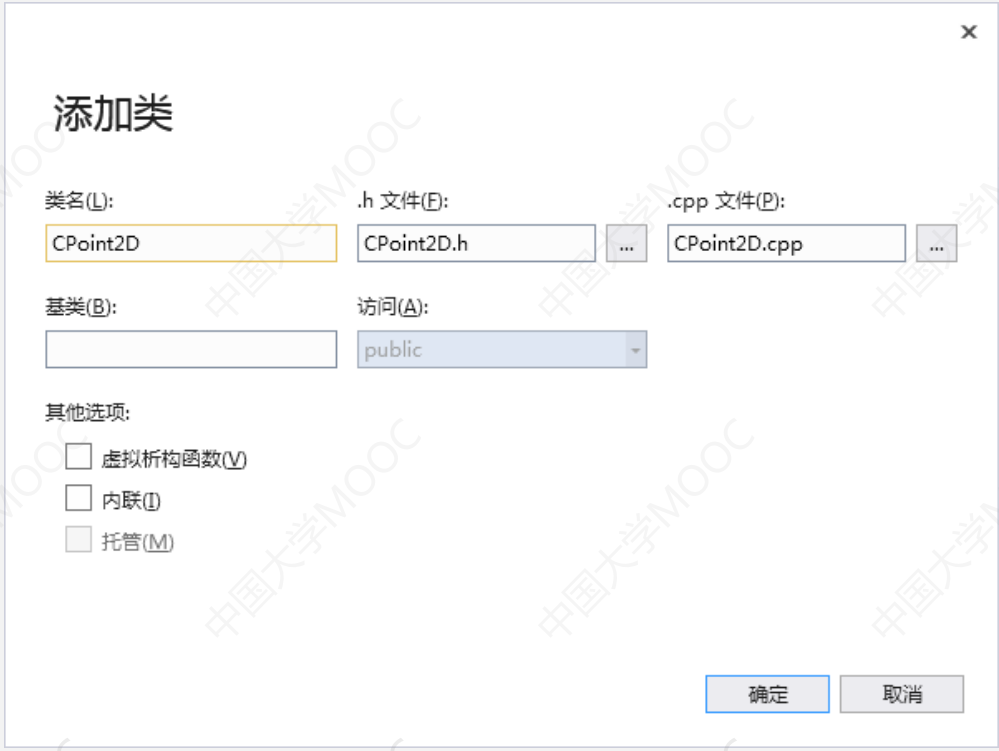
在给定的曲线上保留首末点，每间隔N个点，取一个点。



# 矢量数据压缩算法

Data Compression Algorithms

在Visual Studio 2017平台中，新建一个Windows控制台程序，将该项目命名为VectorDataCompression，首先新建一个类CPoint2D。



# 矢量数据压缩算法

Data Compression Algorithms

在cpp文件里，首先是将idCount设置成1，在构造函数里除了正常的赋上x和y值，此外每增加一个对象，就自动将idCount++，然后将该值赋给m_id，这样确保每个对象都是唯一的。

```
public:
    double GetX() { return m_x; }
    double GetY() { return m_y; }

    void SetX(double x) { m_x = x; }
    void SetY(double y) { m_y = y; }

    int GetID() { return m_id; }

private:
    double m_x;
    double m_y;
    double m_id;

public:
    static int idCount;
};
```

```
int CPoint2D::idCount = 1;

CPoint2D::CPoint2D(double x, double y) :
    m_x(x),
    m_y(y)
{
    m_id = idCount;
    idCount++;
}
```

# 矢量数据压缩算法

Data Compression Algorithms

编写间隔取点法的矢量数据压缩的函数。

```
std::vector<CPoint2D> VectorCompressionByNthPoint(std::vector<CPoint2D> vPoint, int N)
{
    std::vector<CPoint2D> vInervalPoints;
    int vPointSize = vPoint.size();
    for (int loc = 0; loc < vPointSize; )
    {
        vInervalPoints.push_back(vPoint[loc]);
        // 每间隔N个点取一个点
        loc += (N + 1);

        // 如果刚刚到达末尾，取末尾点
        if (loc == vPointSize - 1)
        {
            vInervalPoints.push_back(vPoint[loc]);
            break;
        }
        // 如果超过末尾，将loc设置成末尾点
        if (loc > vPointSize - 1)
        {
            loc = vPointSize -1;
            vInervalPoints.push_back(vPoint[loc]);
            break;
        }
    }
    return vInervalPoints;
}
```

# 矢量数据压缩算法

Data Compression Algorithms

测试一下该函数，首先初始化点集合，该集合由8个点组成。

```
int main()
{
    system("cls");
    cout << "----------------------数据压缩-间隔取点法----------------------" << endl;

    // 初始化点集合
    std::vector<CPoint2D> pts,ptsRes;
    pts.push_back(CPoint2D(1, 1));
    pts.push_back(CPoint2D(2, 3));
    pts.push_back(CPoint2D(3, 2));
    pts.push_back(CPoint2D(4, 5));
    pts.push_back(CPoint2D(5, 1));
    pts.push_back(CPoint2D(6, 8));
    pts.push_back(CPoint2D(7, 2));
    pts.push_back(CPoint2D(8, 3));
```

首先将原始数据打印出来，然后调用VectorCompressionByNthPoint函数，最终打印出来压缩后的结果。

```
cout << "原始数据：" << endl;
for (int i = 0; i < pts.size(); i++)
{
    cout << "(" << pts[i].GetID() << "," << pts[i].GetX() << "," << pts[i].GetY() << ") ";
}
cout << "压缩后数据：" << endl;
ptsRes = VectorCompressionByNthPoint(pts,3);
for (int i = 0; i < ptsRes.size(); i++)
{
    cout << "(" << ptsRes[i].GetID() <<","<< ptsRes[i].GetX() << "," << ptsRes[i].GetY() << ") ";
}
```

# 矢量数据压缩算法

Data Compression Algorithms

测试结果：


再增加一个点，来测试一下

```
// 初始化点集合
std::vector<CPoint2D> pts,ptsRes;
pts.push_back(CPoint2D(1, 1));
pts.push_back(CPoint2D(2, 3));
pts.push_back(CPoint2D(3, 2));
pts.push_back(CPoint2D(4, 5));
pts.push_back(CPoint2D(5, 1));
pts.push_back(CPoint2D(6, 8));
pts.push_back(CPoint2D(7, 2));
pts.push_back(CPoint2D(8, 3));
// 再增加一个点
pts.push_back(CPoint2D(9, 4));
```

# 矢量数据压缩算法

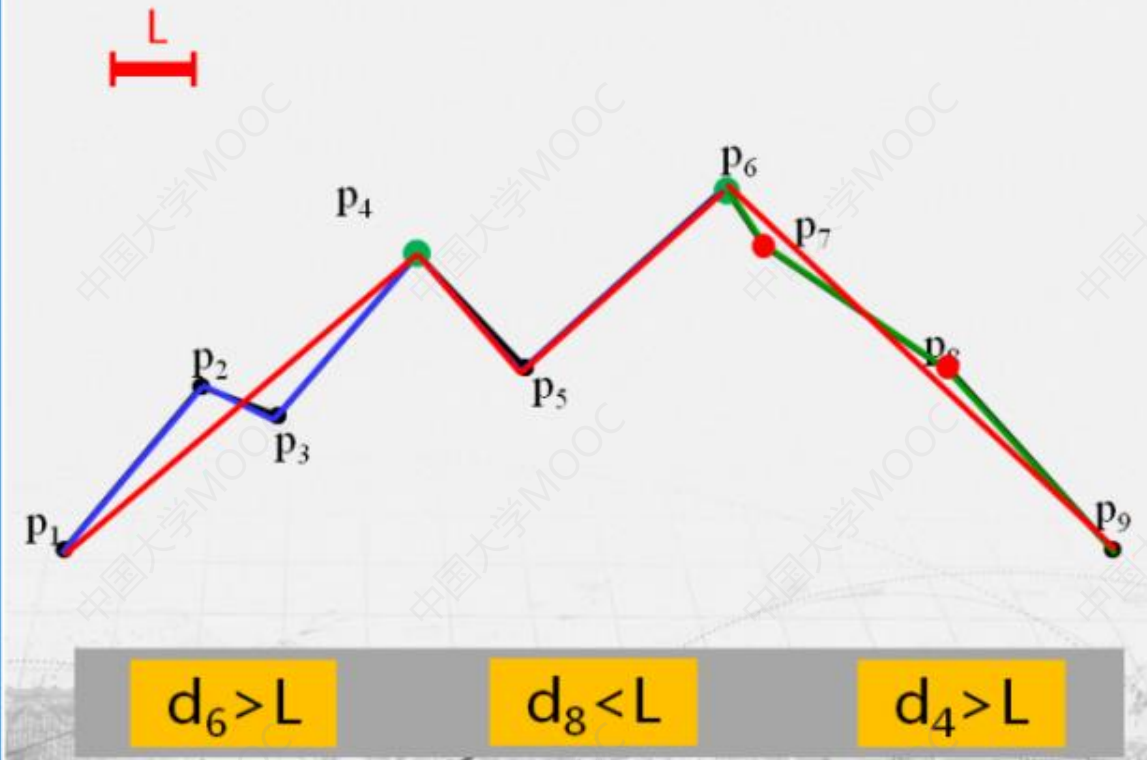
Data Compression Algorithms

## 道格拉斯-普克算法（DP法，David Douglas&Thomas Peucker,1973）

### 算 法 思 路

对给定曲线的首末点连一条直线，求中间所有点与直线间的距离，并找出最大距离dmax，用dmax与限差L比较。

| | |
|---|---|
| $d_{max} \geq L$ | 保留对应点，以该点为界将曲线分为两段，重复使用该方法. |
| $d_{max} < L$ | 舍去所有中间点 |



# 矢量数据压缩算法

Data Compression Algorithms

为了计算点到直线的距离，还需要定义一个直线类，该直线是由两个点组成的，定义了它的成员变量及其访问函数等。

```
class CLine2D
{
public:
    CLine2D(CPoint2D first, CPoint2D second);
    ~CLine2D();
    CPoint2D* GetFristPointPtr() { return &m_Fir; }
    CPoint2D* GetSecondPointPtr() { return &m_Sec; }

private:
    CPoint2D m_Fir;
    CPoint2D m_Sec;
};
```

# 矢量数据压缩算法

Data Compression Algorithms

## 点到直线的距离计算

Ax+By+C=0

点P：$(x_0,y_0)$

$$\frac{|Ax_0+By_0+c|}{\sqrt{A^2+B^2}}$$

```
double PointToLine(CPoint2D point, CLine2D line)
{
    double distance;
    double A, B, C;
    A = -(line.GetSecondPointPtr()->GetY() - line.GetFristPointPtr()->GetY()) / (line.GetSecondPointPtr()->GetX() - line.GetFristPointPtr()->GetX());
    B = 1.0;
    C = -A * line.GetFristPointPtr()->GetX() - line.GetFristPointPtr()->GetY();
    distance = abs(A*point.GetX() + B * point.GetY() + C) / sqrt(A*A + B * B);
    return distance;
}
```

# 矢量数据压缩算法

Data Compression Algorithms

定义函数的输入输出，它的输入是原始的点串，同时给出一个限差L，输出则是压缩后的点串。

```
std::vector<CPoint2D> VectorCompressionByDouglasPeucker(std::vector<CPoint2D>& pointList, float L)
```

首先定义输出的点串变量resultList，然后寻找所有点到首末点的连线最大值。

```
std::vector<CPoint2D> resultList;

// 寻找最大距离点
float dmax = 0;
int index = 0;
CLine2D line(pointList[0], pointList[pointList.size() - 1]);
for (int i = 1; i < pointList.size() - 1; ++i)
{
    double d = PointToLine(pointList[i], line);
    if (d > dmax) {
        index = i;
        dmax = d;
    }
}
```

# 矢量数据压缩算法

Data Compression Algorithms

如果距离大于限差L，以距离最大的点为界，分为两部分，然后再递归调用该算法。

```
//如果距离大于限差
if (dmax > L)
{
    std::vector<CPoint2D> pre_part, next_part;
    for (int i = 0; i <= index; ++i)    pre_part.push_back(pointList[i]);
    for (int i = index; i < pointList.size(); ++i)  next_part.push_back(pointList[i]);
    // 迭代
    std::vector<CPoint2D> resultList1 = VectorCompressionByDouglasPeucker(pre_part, L);
    std::vector<CPoint2D> resultList2 = VectorCompressionByDouglasPeucker(next_part, L);

    // 结合
    resultList.insert(resultList.end(), resultList1.begin(), resultList1.end());
    resultList.insert(resultList.end(), resultList2.begin() + 1, resultList2.end());
}
```

如果距离小于限差L，则直接保留首末点，中间点全部删除，最后返回结果集。

# 矢量数据压缩算法

Data Compression Algorithms

在Main函数里进行测试，仍然是刚才的9个点，提示需要输入限差L的值。然后再调用道格拉斯普克算法，输出结果。

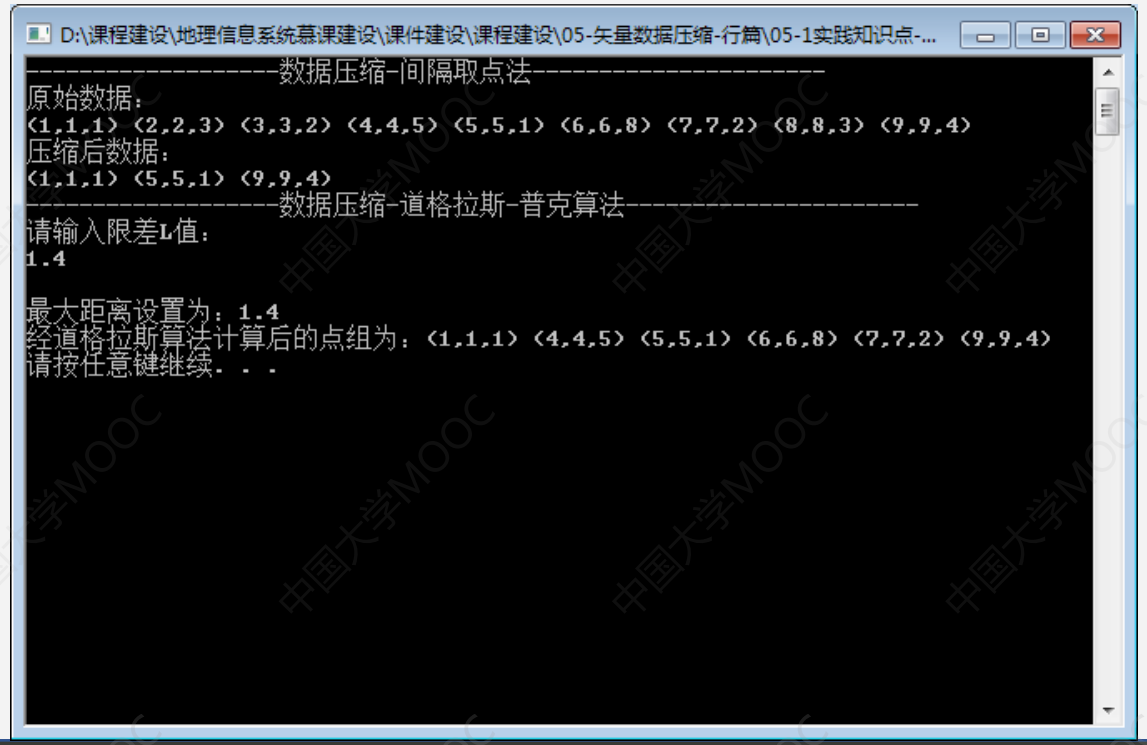
```
cout << "--------------------数据压缩-道格拉斯-普克算法----------------------" << endl;
double L = 1.4;
std::vector<CPoint2D> res = VectorCompressionByDouglasPeucker(pts, L);
cout << "请输入限差L值：" << endl;
cin >> L;
cout << "\n" << "限差L值设置为：" << L << endl;
cout << "经道格拉斯算法计算后的点组为：";
for (int i = 0; i < res.size(); i++)
{
    cout << "(" << res[i].GetID() << "," << res[i].GetX() << "," << res[i].GetY() << ") ";
}
cout << endl;

system("pause");
```

# 矢量数据压缩算法

Data Compression Algorithms

运行一下程序，提示需要输入L值，比如输入L值为1.4，可以得到最终的压缩结果。我们还可以设置不同的限差L来观察这个结果。



# 谢 谢 观 看