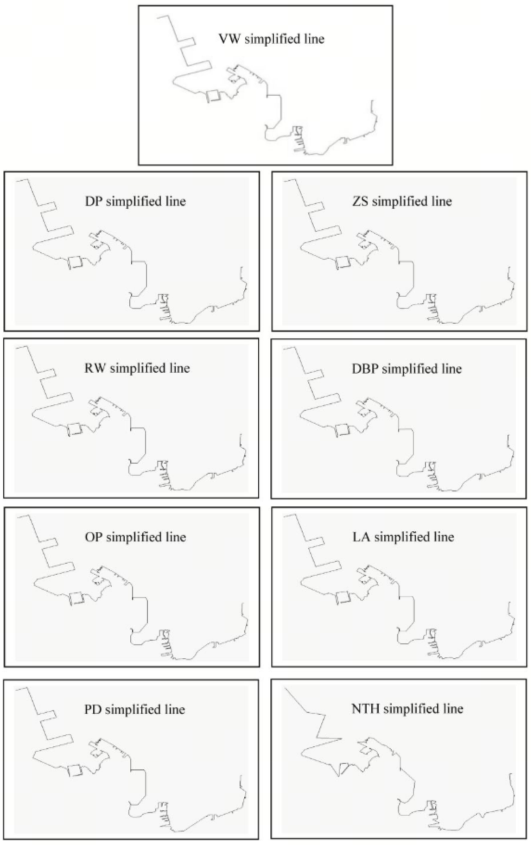
如何优雅地画一条折线（Polyline Simplication）

常见应用：

- 机器人技术中，对旋转式测距扫描仪获取的测距数据进行简化和去噪处理
- 地图中的GPS定位轨迹（水岸、路网）抽稀

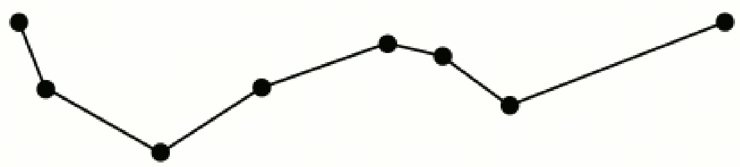


空间时间复杂度

Table of Algorithms

Name	Year	Time	Space	Approximation Factor	Model	Reference
Ramer–Douglas–Peucker algorithm	1972	$O(n^2)$	$O(n)$	Exact	Deterministic	Time
Visvalingam–Whyatt	1993	$O(n^2)$	$O(n)$	Exact	Deterministic	Time
Reumann–Witkam	1974	$O(n)$	$O(1)$	Exact	Deterministic	
Opheim simplification	1981	$O(n)$	$O(1)$	Exact	Deterministic	Time
Lang simplification	1969	$O(n)$	$O(1)$	Exact	Deterministic	
Zhao–Saalfeld	1997	$O(n)$	$O(n)$	Exact	Deterministic	Time

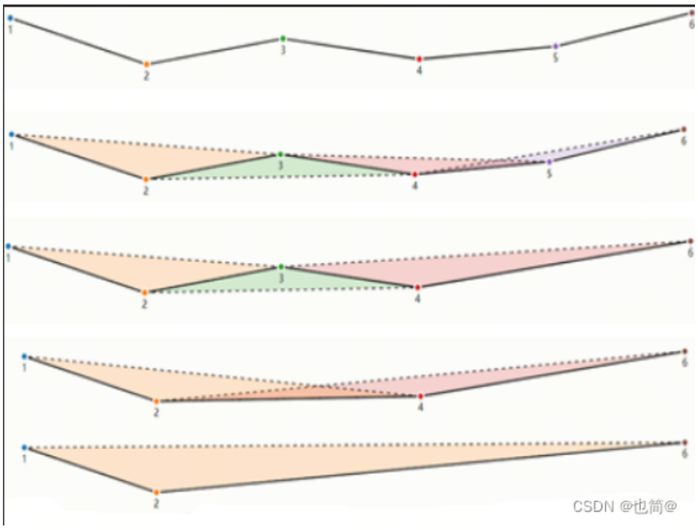
Douglas–Peucker (1972)



```
perpendicularDistance(xy1, xy2, xy) {
  const A = xy2.y - xy1.y;
  const B = xy1.x - xy2.x;
  const C = (xy2.x - xy1.x) * xy1.y - (xy2.y - xy1.y) * xy1.x;

  const distance = Math.abs(A * xy.x + B * xy.y + C) / Math.sqrt(A * A + B * B);
  return distance;
}

DouglasPeucker(_pointList) {
    let _dmax = 0;
    let _index = 0;
    let _length = _pointList.length;
    for (let _i = 1; _i < _length; _i++) {
      let _d = perpendicularDistance(
        _pointList[0],
        _pointList[_length - 1],
        _pointList[_i]
      );
      if (_d > _dmax) {
        _index = _i;
        _dmax = _d;
      }

      let _resultList = [];

      if (_dmax > this.epsilon) {
        const _firstHalf = _resultList.slice(0, _index);
        const _secondHalf = _resultList.slice(_index);
        const _result0 = DouglasPeucker(_firstHalf);
        const _result1 = DouglasPeucker(_secondHalf);

        _resultList = _result0.concat(_result1);
      } else {
        _resultList = [_pointList];
      }
```

```
    }
    return _resultList;
}
```

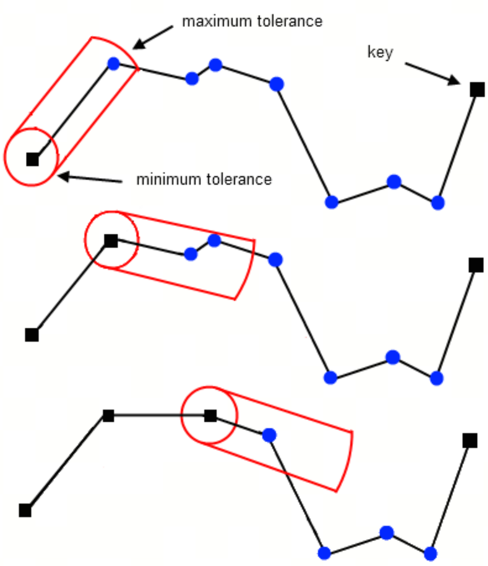
Visvalingam-Whyatt (1993)




```
triangleArea(xy1, xy2, xy3) {
      const x1 = xy1.x;
      const y1 = xy1.y;
      const x2 = xy2.x;
      const y2 = xy2.y;
      const x3 = xy3.x;
      const y3 = xy3.y;

      const area = 0.5 * Math.abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2));
      return area;
};

VisvalingamWhyatt(_pointList) {
let _length = _pointList.length;
let _areas = [];
for (let _i = 0; _i < _length - 2; _i++) {
  let _a = triangleArea(
    _pointList[_i],
    _pointList[_i + 1],
    _pointList[_i + 2]
  );
  _areas.push(_a);
}

while (true) {
```

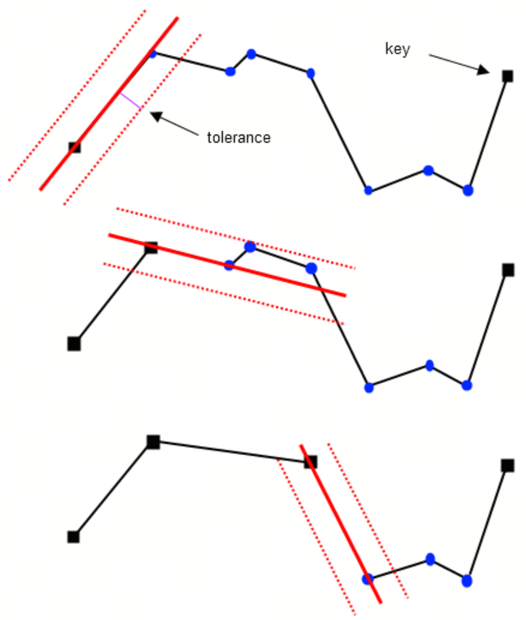
```
    let _minAreaIndex = -1;
    let _minArea = Infinity;

    for (let _i = 1; _i < _length - 1; _i++) {
      if (_areas[_i] < _minArea) {
        _minArea = _areas[_i];
        _minAreaIndex = _i;
      }
    }

    if (_minArea >= this.epsilon) {
      break;
    }

    _pointList.splice(_minAreaIndex, 1);
    _length--;

    if (_minAreaIndex > 1) {
      _areas[_minAreaIndex - 1] = triangleArea(
        _areas[_minAreaIndex - 2],
        _areas[_minAreaIndex - 1],
        _areas[_minAreaIndex]
      );
    }
    if (_minAreaIndex < n - 1) {
      _areas[_minAreaIndex] = triangleArea(
        _areas[_minAreaIndex - 1],
        _areas[_minAreaIndex],
        _areas[_minAreaIndex + 1]
      );
    }
  }

  return _pointList;
}
```

Opheim (1981)



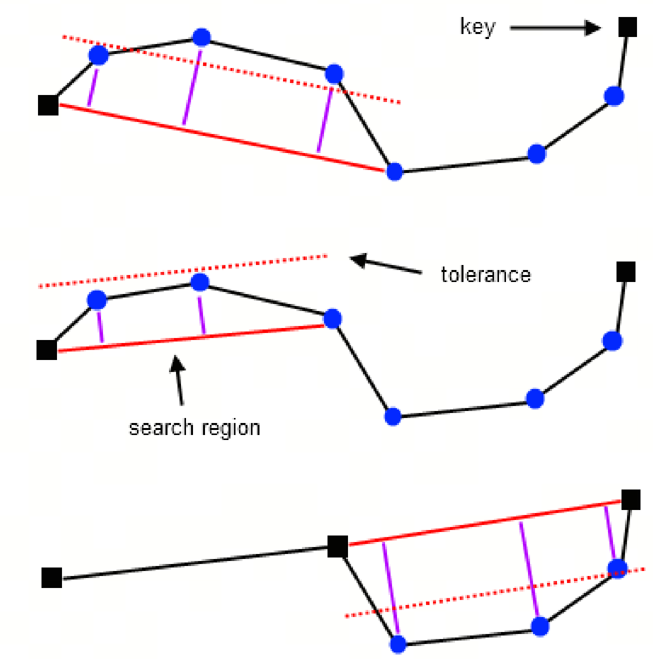

```
isDistanceInRange(xy1, xy2, minDist, maxDist) {
  const dist = Math.sqrt((xy2.x - xy1.x) ** 2 + (xy2.y - xy1.y) ** 2);
  return dist >= minDist && dist <= maxDist;
};

Opheim(_pointList) {
const _minDist = 0.2;
const _maxDist = 1.0;
let _length = _pointList.length;
for (let _i = 0; _i < _length; _i++) {
  let _maxID = _i;
  for (let _j = _i + 1; _j < _length; _j++) {
    let _isInRange = isDistanceInRange(
      _pointList[_i],
      _pointList[_j],
      _minDist,
      _maxDist
    );
    if (_isInRange) {
      _maxID = _j;
    } else {
      break;
    }
  }
  _pointList.splice(_i, _maxID - _i);
}
return _pointList;
}
```

Reumann-Witkam (1974)




```
pointLineDist(xy1, xy2, xy) {
  return (
    Math.abs(
      (xy2.x - xy1.x) * (xy1.y - xy.y) - (xy1.x - xy.x) * (xy2.y - xy1.y)
    ) /
    Math.sqrt(
      (xy2.x - xy1.x) * (xy2.x - xy1.x) + (xy2.y - xy1.y) * (xy2.y - xy1.y)
    )
  );
};

ReumannWitkam(_pointList) {
for (let _i = 0; _i < _pointList.length - 2; _i++) {
  let maxID = _i;
  for (let _j = _i + 2; _j < _pointList.length; _j++) {
    let _dist = pointLineDist(
      _pointList[_i],
      _pointList[_i + 1],
      _pointList[_j]
    );
    if (_dist < this.epsilon) {
      maxID = _j;
    } else {
      break;
    }
  }
  _pointList.splice(_i, maxID - _i);
}
return _pointList;
}
```

Lang (1969)



Zhao-Saalfeld (1997)

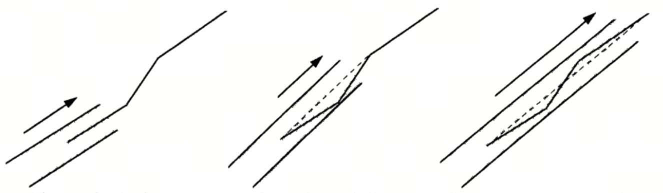


Figure 6. A sleeve moves along the polyline covering consecutive vertices.

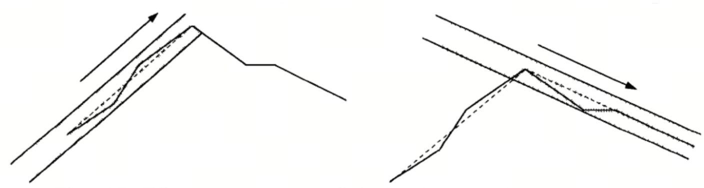


Figure 7. When a vertex cannot fit in the sleeve, a new sleeve is begun

https://cartogis.org/docs/proceedings/archive/auto-carto-13/pdf/linear-time-sleeve-fitting-polyline-simplification-algorithms.pdf

参考链接

https://psimpl.sourceforge.net/documentation.html

Line Simplification (Line Simplification) - Algorithm Wiki

TODO

探索如何和Amaz.LineRenderer配合