WIKIPEDIA

Ramer–Douglas–Peucker algorithm

The **Ramer–Douglas–Peucker algorithm**, also known as the **Douglas–Peucker algorithm** and **iterative end-point fit algorithm**, is an algorithm that decimates a curve composed of line segments to a similar curve with fewer points.

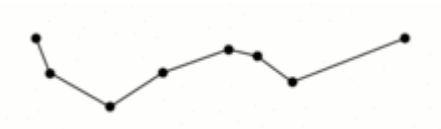
Contents

Idea

The purpose of the algorithm is, given a curve composed of line segments (which is also called a *Polyline* in some contexts), to find a similar curve with fewer points. The algorithm defines 'dissimilar' based on the maximum distance between the original curve and the simplified curve (i.e., the Hausdorff distance between the curves). The simplified curve consists of a subset of the points that defined the original curve.

Algorithm

The starting curve is an ordered set of points or lines and the distance dimension $\varepsilon > 0$.



Simplifying a piecewise linear curve with the Douglas–Peucker algorithm.

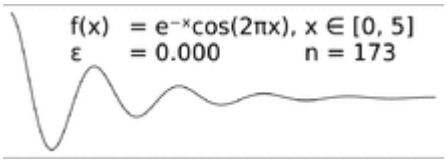
The algorithm recursively divides the line. Initially it is given all the points between the first and last point. It automatically marks the first and last point to be kept. It then finds the point that is farthest from the line segment with the first and last points as end points; this point is obviously farthest on the curve from the approximating line segment between the end points. If the point is closer than ε to the line segment, then any points not currently marked to be kept can be discarded without the simplified curve being worse than ε.

If the point farthest from the line segment is greater than ε from the approximation then that point must be kept. The algorithm recursively calls itself with the first point and the farthest point and then with the farthest point and the last point, which includes the farthest point being marked as kept.

When the recursion is completed a new output curve can be generated consisting of all and only those points that have been marked as kept.

Non-parametric Ramer-Douglas-Peucker

The choice of ε is usually user-defined. Like most line fitting / polygonal approximation / dominant point detection methods, it can be made non-parametric by using the error bound due to digitization / quantization as a termination condition.[1]



The effect of varying epsilon in a parametric implementation of RDP

Pseudocode

(Assumes the input is a one-based array)

```
function DouglasPeucker(PointList[], epsilon)
    // Find the point with the maximum distance
    dmax = 0
    index = 0
    end = length(PointList)
    for i = 2 to ( end - 1) {
        d = perpendicularDistance(PointList[i], Line(PointList[1], PointList[end]))
        if ( d > dmax ) {
            index = i
            dmax = d
        }
    }

    ResultList[] = empty;

    // If max distance is greater than epsilon, recursively simplify
    if ( dmax > epsilon ) {
        // Recursive call
        recResults1[] = DouglasPeucker(PointList[1...index], epsilon)
        recResults2[] = DouglasPeucker(PointList[index...end], epsilon)

        // Build the result list
        ResultList[] = {recResults1[1...length(recResults1)-1], recResults2[1...length(recResults2)]}
    } else {
        ResultList[] = {PointList[1], PointList[end]}
    }
    // Return the result
    return ResultList[]
end
```

link : https://karthaus.nl/rdp/

Application

The algorithm is used for the processing of vector graphics and cartographic generalization. It does not always preserve the property of non-self-intersection for curves which has led to the development of variant algorithms[2].

The algorithm is widely used in robotics[3] to perform simplification and denoising of range data acquired by a rotating range scanner; in this field it is known as the split-and-merge algorithm and is attributed to Duda and Hart.[4]

Complexity

The expected complexity of this algorithm can be described by the linear recurrence $T(n) = 2T(n/2) + O(n)$, which has the well-known solution (via the master theorem for divide-and-conquer recurrences) of $T(n) \in \Theta(n \log n)$. However, the worst-case complexity is $\Theta(n^2)$.

Other line simplification algorithms

Alternative algorithms for line simplification include:

- Visvalingam–Whyatt
- Reumann–Witkam
- Opheim simplification
- Lang simplification
- Zhao-Saalfeld

Further reading

- Urs Ramer, "An iterative procedure for the polygonal approximation of plane curves", Computer Graphics and Image Processing, 1(3), 244–256 (1972) doi:10.1016/S0146-664X(72)80017-0 (https://doi.org/10.1016%2FS0146-664X%2872%2980017-0)
- David Douglas & Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer 10(2), 112–122 (1973) doi:10.3138/FM57-6770-U75U-7727 (https://doi.org/10.3138%2FFM57-6770-U75U-7727)
- John Hershberger & Jack Snoeyink, "Speeding Up the Douglas–Peucker Line-Simplification Algorithm", Proc 5th Symp on Data Handling, 134–143 (1992). UBC Tech Report TR-92-07 available at http://www.cs.ubc.ca/cgi-bin/tr/1992/TR-92-07
- R.O. Duda and P.E. Hart, "Pattern classification and scene analysis", (1973), Wiley, New York (https://web.archive.org/web/20110715184521/http://rii.ricoh.com/~stork/DHS.html)
- Visvalingam, M; Whyatt, JD (1992). *Line Generalisation by Repeated Elimination of the Smallest Area* (https://hydra.hull.ac.uk/assets/hull:8338/content) (Technical report). Discussion Paper. Cartographic Information Systems Research Group (CISRG), The University of Hull. 10.

References

1. Prasad, Dilip K.; Leung, Maylor K.H.; Quek, Chai; Cho, Siu-Yeung (2012). "A novel framework for making dominant point detection methods non-parametric". *Image and Vision Computing*. **30** (11): 843–859. doi:10.1016/j.imavis.2012.06.010 (https://doi.org/10.1016%2Fj.imavis.2012.06.010).
2. Wu, Shin-Ting; Marquez, Mercedes (2003). "A non-self-intersection Douglas-Peucker algorithm" (https://ieeexplore.ieee.org/document/1240992). *16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003)*. Sao Carlos, Brazil: IEEE. pp. 60–66. CiteSeerX 10.1.1.73.5773 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.5773). doi:10.1109/SIBGRA.2003.1240992 (https://doi.org/10.1109%2FSIBGRA.2003.1240992). ISBN 978-0-7695-2032-2.
3. Nguyen, Viet; Gächter, Stefan; Martinelli, Agostino; Tomatis, Nicola; Siegwart, Roland (2007). *A comparison of line extraction algorithms using 2D range data for indoor mobile robotics* (http://doc.rero.ch/record/320492/files/10514_2007_Article_9034.pdf) (PDF). *Autonomous Robots*. **23** (2). pp. 97–111. doi:10.1007/s10514-007-9034-y (https://doi.org/10.1007%2Fs10514-007-9034-y).
4. Duda, Richard O.; Hart, Peter E. (1973). *Pattern classification and scene analysis* (https://archive.org/details/patternclassific0000duda). New York: Wiley. ISBN 0-471-22361-1.

External links

- Boost.Geometry support Douglas–Peucker simplification algorithm (https://www.boost.org/doc/libs/1_67_0/libs/geometry/doc/html/geometry/reference/algorithms/simplify/simplify_3.html)
- Implementation of Ramer–Douglas–Peucker and many other simplification algorithms with open source licence in C++ (http://www.codeproject.com/Articles/114797/Polyline-Simplification)
- XSLT implementation of the algorithm for use with KML data. (http://idea.ed.ac.uk/data/kmz/)
- You can see the algorithm applied to a GPS log from a bike ride at the bottom of this page (http://www.bdcc.co.uk/Gmaps/Services.htm)
- Interactive visualization of the algorithm (http://karthaus.nl/rdp/)
- Implementation in F# (http://fssnip.net/kY)

- Ruby gem implementation (https://github.com/odlp/simplify_rb)
- JTS, Java Topology Suite (https://github.com/locationtech/jts), contains Java implementation of many algorithms, including the Douglas-Peucker algorithm (https://locationtech.github.io/jts/javadoc/org/locationtech/jts/simplify/DouglasPeuckerSimplifier.html)
