

Representations, Structures, and Algorithms

1 Literature Review

Paper 1: *Exploring multiple viewshed analysis using terrain features and optimisation techniques [1]*

Summary

Foundational viewshed algorithms covered in this course determine what areas may be seen from a given point (viewshed analysis) or if one point can be seen from another point (intervisibility analysis). Viewshed analysis also supports sophisticated terrain-based modeling, such as optimisation of multiple observer locations on a terrain.

Despite the utility of such analysis, precise algorithms to accomplish this optimisation remain computationally intensive for non-trivial cases. Due to the combinatorial nature of this problem, locating v viewpoints directly within an area of interest of n pixels has complexity $\mathcal{O}(nv)$. This paper reduces this runtime through two methods: reduction of the candidate viewpoint set and adoption of appropriate heuristics for multiple location selection. The authors demonstrate significantly reduced computation time at the cost of $\sim 10\%$ reduction in visible area.

Algorithm and Software

Analysis for this paper was undertaken using Landserf, a free GIS software developed through the mid-2000s. Three main algorithms were used: viewshed analysis, surface classification, and heuristic algorithms. Viewshed analysis was implemented as discussed in class.

Polynomial trend surfaces were generated in Landserf to classify every point within the DEM as a pit, valley, pass, ridge, peak, or planar slope based on the coefficients of the polynomial derived from the 9x9 pixel neighbourhood.

Three heuristic algorithms were chosen for viewpoint selection. The swap algorithm greedily improves the starting solution by switching out a viewpoint in the solution for another point and comparing outputs. The spatial genetic algorithm draws inspiration from evolutionary processes to “breed” a final solution. Finally, the spatial simulated annealing algorithm draws inspiration from metallurgy to gradually settle into an optimal solution.

Data and Data Source

This paper used data from the Ordnance Survey PANORAMA DEM and obtained from the ED-INA Digimap service supplied by JISC. The 20x20km analysed covers the Cairngorm mountain area of Scotland and was originally provided at 50m resolution.

Results

Reduction of the candidate viewpoint set reduced processing time by two orders of magnitude and final viewshed area by up to 10% for two to ten observers compared with processing all candidate pixels.

Heuristics adapted from location allocation literature reduced the runtime from ~ 56 h to ~ 0.32 h-8.26h. Algorithms were compared amongst one another and found to provide similar outputs but varying runtimes, with the swap algorithm offering the best overall performance.

Data and Algorithm Quality Issues

The data was downsampled from 50m to 500m to provide a small subject area with significant terrain variation. It is unclear what impact this had on the results or if key landscape characteristics were maintained.

Candidate viewpoint reduction methods were compared using a heuristic method. It was not discussed which heuristic was used nor whether some heuristics might exhibit biases for or against particular morphometric point types. Since a direct solution could not be computed for cases with more than two observers, it is impossible to compare the output from the heuristics with a objective solution.

Paper 2: *An efficient GPU multiple-observer siting method based on sparse-matrix multiplication* [2]

Summary

Optimal siting of multiple observers on an arbitrary terrain is NP hard and cannot be computed directly. Heuristics such as swap search provide an approximate answer yet still require tremendous processing resources and cannot cope with the increasing amount of high resolution terrain data now available. To mitigate these concerns, the authors devised a method for multiple observer siting taking advantage of parallelization techniques and increasingly available GPU software and hardware (here, NVIDIA’s CUDA). The algorithm developed, *SparseSite*, selects observers on a terrain that can collectively visualize a given percent of the study area. They focus on three objectives for improvement: decreasing observers required to reach this desired coverage, reduction in processing resources, and robustness for large datasets. *SparseSite* successfully operates on large terrains (150002 cells) with comparable to reduced time and memory requirements to existing parallel implementations.

Algorithm and Software

Software for *SparseSite* was implemented in C++/CUDA, compiled using g++ 4.8 and nvcc 4.0. It was designed for use on CUDA enabled NVIDIA GPUs. *SparseSite* builds off existing viewshed and observer siting algorithms (e.g. *SiteGSM*) which use a swap heuristic. Generally, this algorithm begins with a set of observers which are “swapped” out for observers outside the set to determine which leads to the greatest increase in total visible area. *SparseSite* offers three important improvements.

First, a local search is included in the swap, reducing the total number of observers required. Second, the swap heuristic is implemented using dynamic programming methods uniquely suited to sparse-dense matrix multiplication, avoiding the costly re-computation of joint visibility indices. Finally, memory use is reduced due to this dynamic programming approach.

Data and Data Source

SparseSite was tested on 30m resolution elevation data from the state of Illinois with four cell quantities: 12012, 36012, 75002, and 150002. Different size terrains allowed for comparison of performance across small and large terrains as runtime and memory use may be non-linear for this algorithm. Data was downloaded from the NASA SRTM project.

Results

Since traditional commercial and open source GIS software (e.g. ArcGIS, QGIS, GRASS) offer only viewshed analysis from known locations, results are compared against existing sequential and parallel implementations of the swap algorithm (*Site* and *SiteGSM* respectively).

First, *SparseSite* was able to process large terrains that could not be addressed using other methods. Second, in most cases *SparseSite* was faster than existing methods (up to 2.7x), particularly when the radius of interest was large. Finally, local search reduced the number of observers required for a given coverage by $\sim 10\%$.

Data and Algorithm Quality Issues

SparseSite accounts for redundant computation and the pitfalls of a greedy search heuristic. It does not account for viewshed uncertainty or partial (non-boolean) visibility, a general weakness in viewshed analysis.

No discussion of data selection is included, and results are assumed to generalize to any similar terrain dataset. The dataset has known inconsistencies (voids, null areas) which were not specifically addressed.

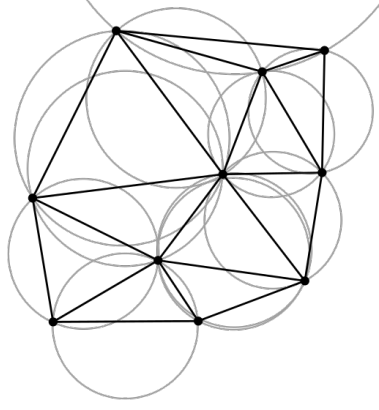


Figure 1: Depiction of a Delaunay triangulation (black) and its circumcircles (grey) [3].

2 GIS Comparison

Introduction

To peer under the hood of a GIS requires considerable time and effort. For most commercial GIS, such as ArcGIS, investigating the mechanisms of the algorithms directly is all but impossible. Nevertheless, details of the implementation of these algorithms impacts the value and quality of the results they provide. An active GIS user must be aware of these pitfalls and work to directly mitigate any known weaknesses or edge cases.

The following analysis investigates two such algorithms: Delaunay triangulation, a staple for TIN creation, and line simplification. For each algorithm we provide basic theory about its general steps as well as details about its implementation in ArcGIS and QGIS as available. Next, we identify potential weak points and develop datasets to test edge cases. We present and discuss the outcomes of these tests. Finally, we conclude with suggestions for further research and a note of caution for GIS users.

Background

1 Delaunay Triangulation

A Delaunay triangulation for arbitrary dimension d is defined in [4] as follows.

Definition. A Delaunay triangulation \triangle of a set of points \mathbf{P} in d dimensions if the open discs (balls) circumscribed to any of its elements does not contain any point within \mathbf{P} .

In a two dimensional space, such as a geographic plane, the Delaunay criterion is satisfied if and only if no circle circumscribing a triangle contains another point in the set. This is illustrated in Figure 1.

A Delaunay triangulation does not exist if all points are coincident. In addition, a Delaunay triangulation may not be unique [4].

Theorem. A Delaunay triangulation \triangle of set of points \mathbf{P} in d dimensions is not guaranteed to be unique if $d+2$ points of \mathbf{P} are situated on the boundary of an open disc (ball) b , where b contains no other points (that is, where points are in the general position).

A number of algorithms for determining the Delaunay triangulation (\triangle) of a set of points exist. Since \triangle is the dual graph of a Voronoi diagram, it can be derived directly from Thiessen polygons. This method is necessarily restricted to 2D. Iterative methods allow \triangle to be built incrementally by selecting a baseline edge from the Convex hull, then adding points as appropriate. Divide and conquer methods break the point set into smaller sets, compute the triangulation separately, and then rejoin [5]. These two methods can be generalized to arbitrary dimensions. In general, the runtime of these algorithms is $\mathcal{O}(n \log n)$. Unfortunately no details are available for the ArcGIS implementation of this method, nor is there QGIS documentation [6, 7]. QGIS source code for these functions can be found at [8, 9].

From the above theorem, it is clear that the handling of points in the general position may be a source of ambiguity or failure. In addition, results may differ between triangulations computed in 2D vs 3D.

2 Douglas-Peucker Line Simplification

While a number of algorithms exist for line simplification, the most commonly used is Douglas-Peucker due to its speed ($\mathcal{O}(n \log n)$) and simplicity. This method retains key points of the original line and ensures that all original points lie within some user-specified tolerance $\epsilon > 0$ of the simplification [10]. All points and line segments in the simplified line necessarily lie within the convex hull of the original line.

This recursive algorithm consists of the following steps for start points a and b (initially the start and end points of the line):

1. Draw trendline \vec{ab} between a and b .
2. Select vertex c farthest from the trendline where the distance is at least ϵ .
3. Add c to the final set of vertices, creating line segments \vec{ac} and \vec{cb} .
4. Simplify \vec{ac} and \vec{cb} .

As noted in [11], there are at least two cases where this algorithm will fail to produce an appropriate result.

First, points lying within the convex hull of the simplified segments may be displaced from one side of the line to the other. In practice, a landmark located on the North Bank of the Thames River may be located on the South Bank of the simplified Thames River polyline.

Second, topological consistency of the line may not be maintained. In cases where vertices of one subpolyline intersect the convex hull of another subpolyline (that is, the convex hulls of two subpolylines overlap), the resultant simplified line may intersect itself [11, 12]. This typically occurs where ϵ is relatively large.

Improvements to this algorithm have sought to mitigate this edge point by reducing ϵ for vertices within the self-crossing segment [11] or modifying the vertices selected to other vertices not within the original set [12].

In this analysis, we investigate the second of these cases.

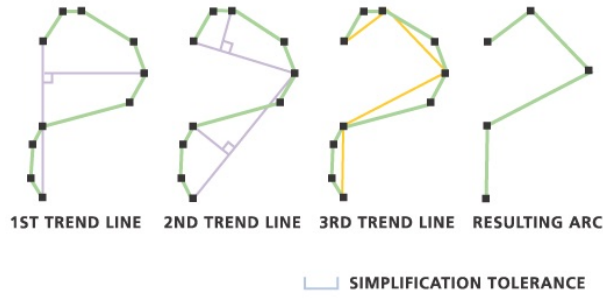


Figure 2: The general progression of the Douglas-Peucker algorithm [13].

Data

1 Delaunay Triangulation

To test whether the Delaunay triangulation was performed in 2D or 3D, we created an artificial dataset with several sets of circular points stacked vertically. At the base of this stack were two additional rings of points with increasing diameter, creating a shape reminiscent of a stovepipe hat (see Figure 4). This was created in Python as a CSV file with no intended CRS. Elevation (z value) ranged from 100 to 2000.

To test how the general case (i.e. four points on a circle) was handled, we created seven CSV files each containing four equidistant points on a circle (i.e. a square) rotated about its centre in 15° intervals from 0° to 90° . Points diagonal from one another were given the same z values, either 100 or 200. This configuration of points has two possible Delaunay triangulations. One connects the points with high altitudes, resulting in a ridge, while the other connects the points with low altitudes, resulting in a valley. CSVs were created in R with no intended CRS.

2 Douglas-Peucker Line Simplification

Two lines were hand drawn in ArcGIS based on diagrams provided in [11]. Although they were created in BNG within the boundaries of London, they do not represent any existing or notional spatial content and are intended only as standalone examples.

Method

1 Delaunay Triangulation

In ArcGIS, Delaunay triangulation is performed as part of the ‘Create TIN’ tool. In QGIS, Delaunay triangulation may be computed explicitly using the ‘Delaunay triangulation’ tool or implicitly by interpolating points to a raster via TIN method.

To determine if any of these triangulation methods were done in 3D, we imported the CSV into each GIS as points with CRS BNG then ran the tools listed above. In QGIS, the interpolated raster was used to determine which level of points within the vertical stack was used (and which were discarded). This process was repeated using WGS84 as the CSV CRS.

If the algorithm performed the triangulation in 3D, the resulting TIN should have a sharp right angle at the base of the circular stack. If not, we would expect to see a flat plane or a sloped

pilgrim hat shape, depending on the centre points selected.

2 Douglas-Peucker Line Simplification

In ArcGIS, the ‘Simplify Line’ tool utilizes a form of the Douglas-Peucker algorithm and includes a user-defined parameter for threshold distance [13]. Users may elect to “check for topological errors”, though this simply appends a boolean field within the attribute table indicating the presence of topological issues.

In QGIS, two tools use a form of Douglas-Peucker: ‘simplify geometries’ and ‘generalize.simplify’ (within GRASS).

The shapefile was loaded in each GIS and simplified using the above tools to test if any topological errors occurred. The threshold distance was adjusted to elicit potential line crossing behaviour. To compare performance in WGS84, the file was reprojected within each GIS and the process repeated.

Results

1 Delaunay Triangulation

As shown in Figure 3 on page 8, methods in both QGIS and ArcGIS generated a sloping pilgrim hat triangulation using only the topmost centre points. 3D views of the points and resultant triangulation are shown in Figure 4 on page 8. Triangulation of the circle at the top of the pilgrim hat differed slightly between the two systems. No difference was observed in triangulation between BNG and WGS84.

As shown in Figure 5 on page 9, results from the QGIS TIN interpolation tool and the ArcGIS Create TIN tool were consistent. In both cases, the diagonal line dividing the square into triangles connected the lowest points, resulting in a valley. Interestingly, these results differed from the triangulation obtained using the QGIS Delaunay Triangulation tool. While squares rotated by 0° , 15° , and 75° exhibit triangulation consistent with the TIN interpolation tool, squares with rotation 30° , 45° , 60° , and 90° instead connect the highest points, resulting in a ridge. (Note that this results in the same triangulation for 0° and 90° .)

No differences between BNG and WGS84 were present in these results.

2 Douglas-Peucker Line Simplification

Results are summarized in Figure 6 on page 10. All lines created using the ArcGIS ‘Simplify Line’ tool had self-intersections. No topological or structural differences were observed for either line in WGS84 vs BNG.

Simplification in BNG using the QGIS/GRASS ‘Generalisation.Simplification’ tool resulted in one and two self intersections for Line 1 and Line 2 respectively. In contrast, *neither* line created using the QGIS ‘Simplify Geometries’ tool had any self intersections in BNG or WGS84.

Interestingly, results for Line 1 differed significantly between WGS84 and BNG for both simplification methods. The Simplify Geometries method resulted in a line with an additional node, while the Generalisation method resulted in a line with two self-intersections (instead of one in BNG).

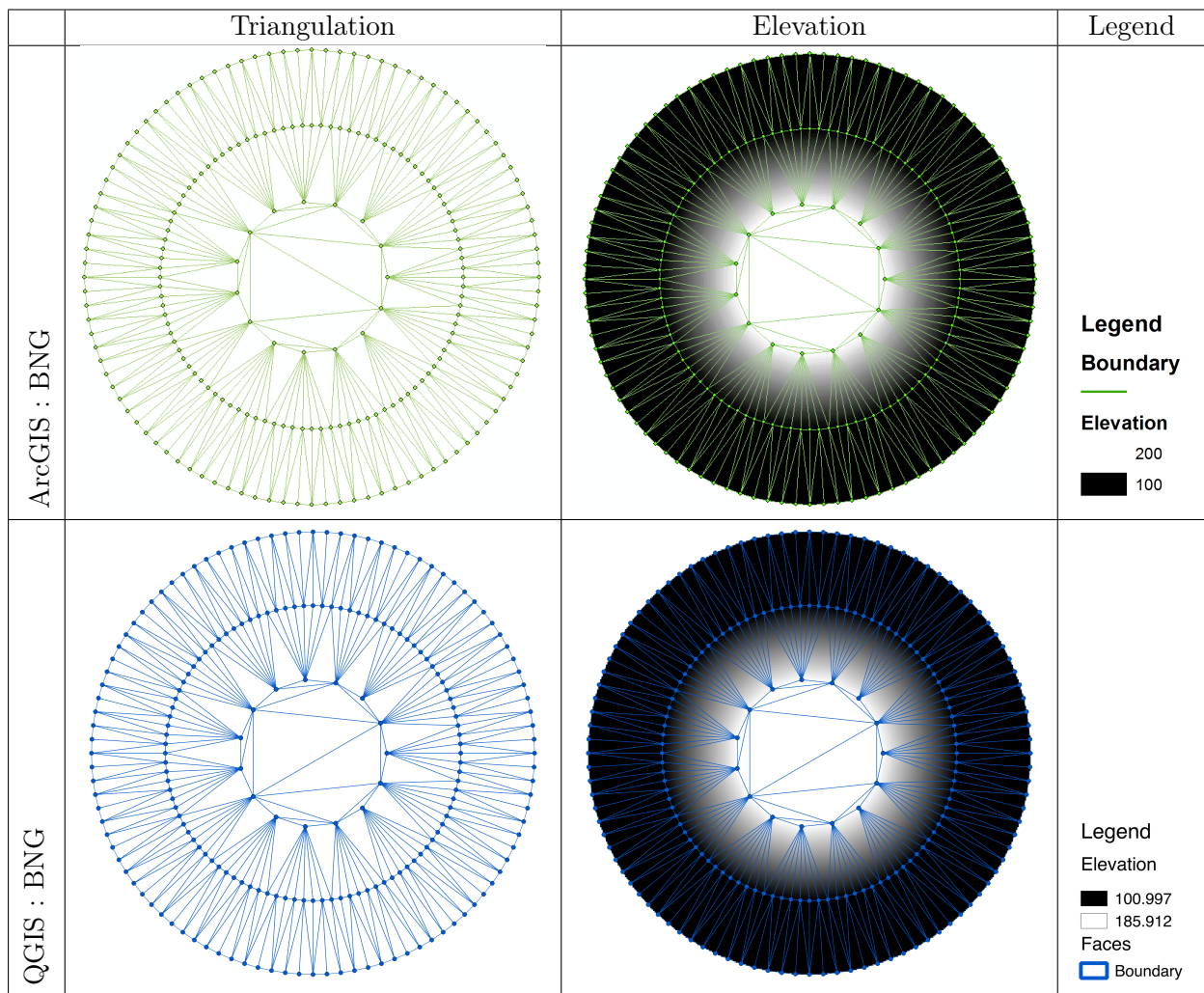


Figure 3: Comparison of triangulations of the “hat” test case in BNG in both GIS.

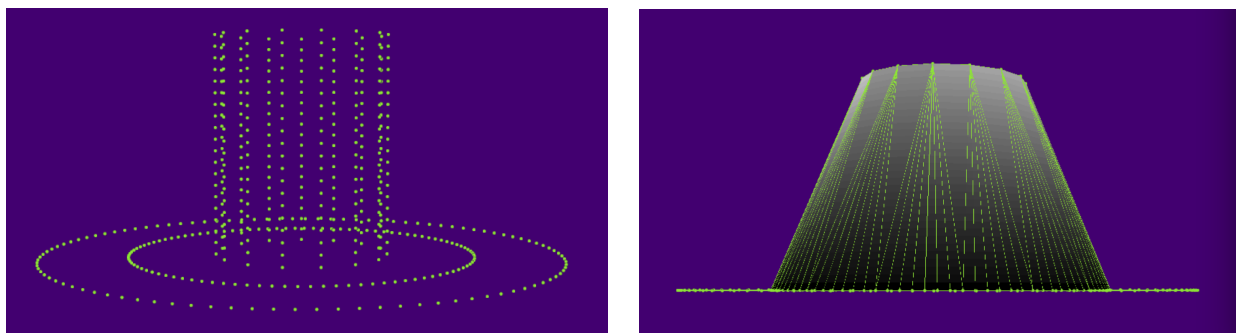


Figure 4: 3D views of the “hat” test case points and resultant TIN in ArcScene.

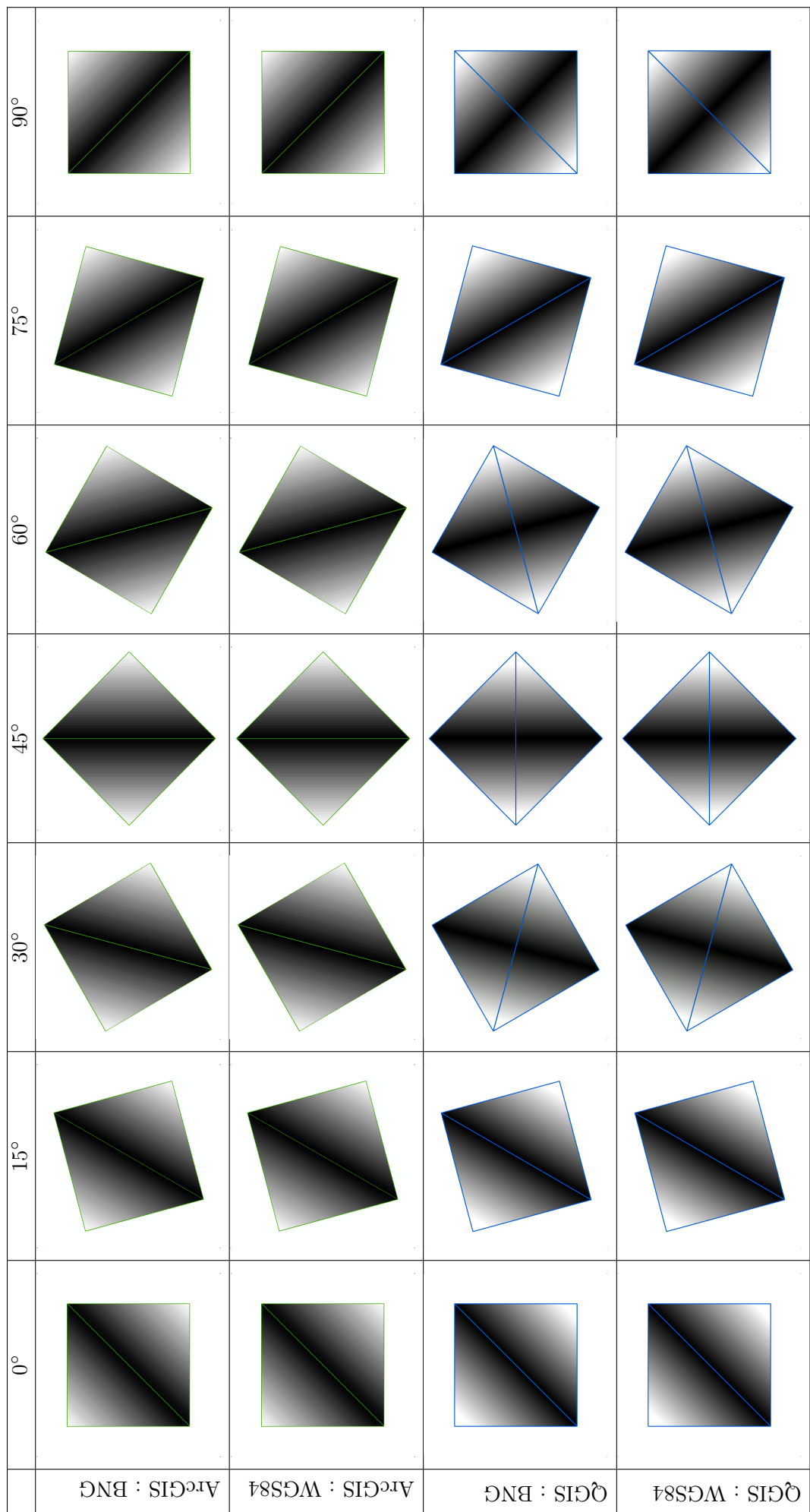


Figure 5: Results of square rotation showing TIN elevation (black = low, white = high) and Delaunay triangulations (green and blue) in ArcGIS and QGIS in both BNG and WGS84.

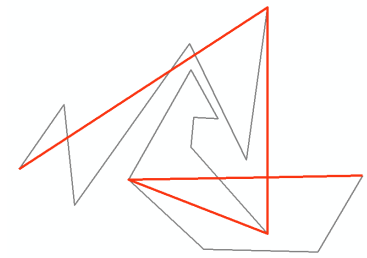
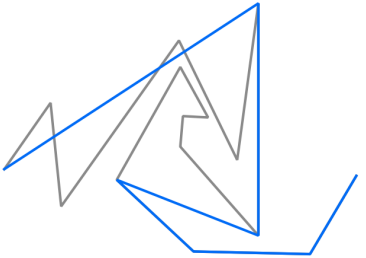
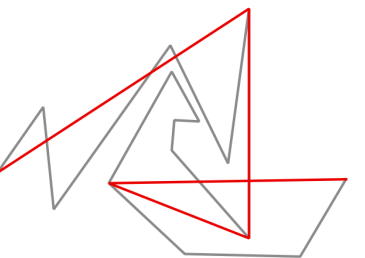
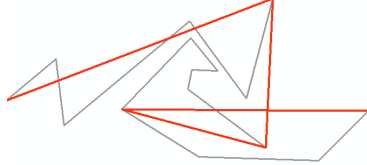
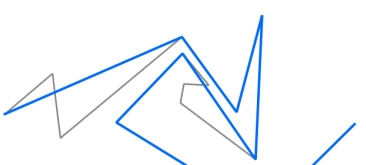

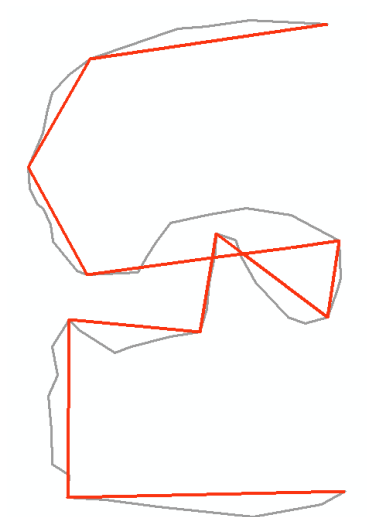
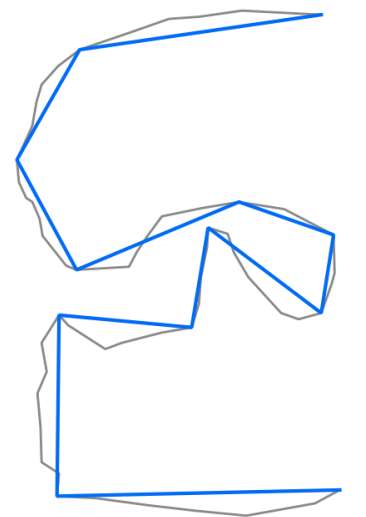
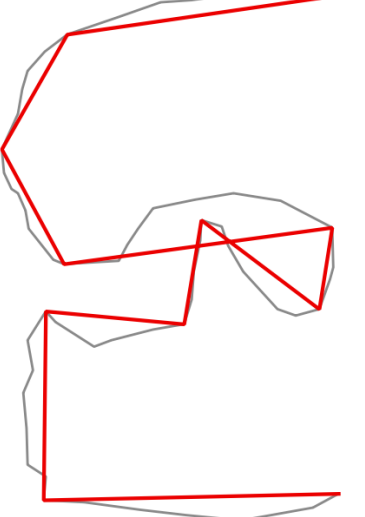
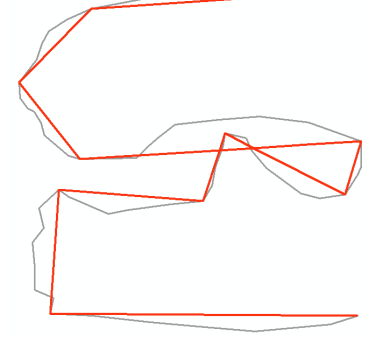
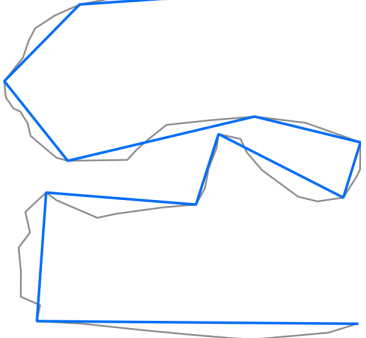
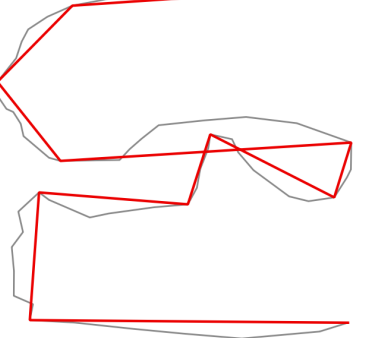
	ArcGIS - Simplify Line	QGIS - Simplification	QGIS - Generalisation
Line 1 : BNG			
Line 1 : WGS84			
Line 2 : BNG			
Line 2 : WGS84			

Figure 6: A table showing results for two lines that result in self-crossing simplifications, comparing analysis in ArcGIS and QGIS as well as BNG and WGS84. The original line is shown in grey and algorithm outputs are shown in red (self-intersecting) or blue (not self-intersecting).

Discussion

1 Delaunay Triangulation

The “hat” case yielded expected results, confirming that both ArcGIS and QGIS perform triangulation in 2D. While this is likely faster than a 3D implementation, it may obfuscate underlying trends and can be easily skewed by high vertical outliers. GRASS GIS offers a 3D alternative [14].

Results for the general (square) case were more surprising. In cases where two or more Delaunay triangulations are equally valid, such as the four corners of a square, algorithms must nevertheless select only one triangulation. Such a selection, though arbitrary, is likely to be made consistently by each implementation. For all rotations studied, both ArcGIS and QGIS TIN generation prioritize a valley. The QGIS Delaunay Triangulation tool resulted in both valley and ridge outcomes, suggesting that elevation may not be the discriminating factor between valid triangulations. Additional datasets, such as flat or sloped squares, should be used to further test this edge case.

Changing CRS had no impact on triangulation for these small examples, as expected.

2 Douglas-Peucker Line Simplification

Performance for this algorithm was not as robust as expected. Both ArcGIS Simplify Line and QGIS/GRASS Generalize resulted in self intersections. However, QGIS Simplify had *no* self intersections for either case in either CRS. A quick look at the source code indicates that the later is designed explicitly to preserve topology ([15, line 35]) whereas no such requirement is present in the former ([16]).

Additional edge cases, such as point displacement from one side of a line to another, were not tested.

The difference in performance in QGIS between BNG and WGS84 are likely attributable to a difference in the threshold distance provided, as the units were in meters and decimal degrees respectively. This may be an error by the author.

Conclusion and Further Work

While this analysis presents minimum cases where the algorithms fail to deliver an appropriate or expected result, similar results are likely to occur in real world data. Further tests are required to determine the extent of this impact of these algorithmic weaknesses on a larger integrated example.

As always, the ultimate responsibility for accuracy of the final product relies on GIS users, not the software itself. As Dr. George Jenks noted in *Lines, Computers, and Human Frailties*, “Man-induced errors may exceed, indeed overshadow, errors attributable to equipment” [17]. Results must always be considered critically in light of the algorithms and software used and the quality of the data provided.

References

- [1] Kim, Y.H., Rana, S. and Wise, S., 2004. Exploring multiple viewshed analysis using terrain features and optimisation techniques. *Computers & Geosciences*, 30(9), pp.1019-1032.
- [2] Pena, G.C., Magalhães, S.V., Andrade, M.V., Franklin, W.R., Ferreira, C.R. and Li, W., 2014, November. An efficient GPU multiple-observer siting method based on sparse-matrix multiplication. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data* (pp. 54-63). ACM.
- [3] https://upload.wikimedia.org/wikipedia/commons/thumb/d/db/Delaunay_circumcircles_vectorial.svg/560px-Delaunay_circumcircles_vectorial.svg.png [image]
- [4] Frey, P.J. and George, P.L., 2000. *Mesh generation: application to finite elements* p.20. Paris: Hermes Science.
- [5] Bearman, N. 2017, *Fields*, lecture notes, Representations Structures and Algorithms GE-OGG126 UCL, delivered 23 Nov 2017.
- [6] Anon, Fundamentals of TIN triangulation in ArcGIS—Help. *ArcGIS for Desktop*. Available at: <http://desktop.arcgis.com/en/arcmap/10.3/manage-data/tin/fundamentals-of-tin-triangulation.htm> [Accessed 11 Jan 2018].
- [7] Anon, Vector Geometry. *Documentation QGIS 2.18*. Available at: https://docs.qgis.org/2.18/en/docs/user_manual/processing_algs/qgis/vector_geometry_tools.html?highlight=delaunay#delaunay-triangulation [Accessed 11 Jan 2018].
- [8] Olaya, V. 2012. *Delaunay.py QGIS* (master). [source code] Available at: <https://github.com/qgis/QGIS/blob/master/python/plugins/processing/algs/qgis/Delaunay.py> [Accessed 12 Jan 2018].
- [9] Bruy, A. 2016. *TinInterpolation.py QGIS* (master). [source code] Available at: <https://github.com/qgis/QGIS/blob/master/python/plugins/processing/algs/qgis/TinInterpolation.py> [Accessed 12 Jan 2018].
- [10] Dowman, I. 2017, *Cartographic Design and Generalisation*, lecture notes, Mapping Science GEOGG034, delivered, delivered Nov 2017.
- [11] Saalfeld, A., 1999. Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm. *Cartography and Geographic Information Science*, 26(1), pp.7–18.
- [12] Wu, S.T. and Marquez, M.R.G., 2003, October. A non-self-intersection Douglas-Peucker algorithm. In *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on* (pp. 60-66). IEEE.
- [13] Anon, How Simplify Line or Polygon (Coverage) works—Help. *ArcGIS for Desktop*. Available at: <http://desktop.arcgis.com/en/arcmap/10.3/tools/coverage-toolbox/how-simplify-line-or-polygon-works.htm> [Accessed 11 Jan 2018].
- [14] Anon, Creating Tin from Elevation Points in QGIS? *Geographic Information Systems Stack Exchange*. Available at: <https://gis.stackexchange.com/questions/150291/creating-tin-from-elevation-points-in-qgis> [Accessed 12 Jan 2018].
- [15] Kuhn, M. and Huarte, A. 2013. *qgissimplifymethod.j QGIS* (master) [source code] Available at: <https://github.com/qgis/QGIS/blob/master/src/core/qgssimplifymethod.h> [Accessed 12 Jan 2018]

- [16] Bundala, D. 2005. v.Generalize. *GRASS GIS* (v7.2). [source code]. Available at: http://svn.osgeo.org/grass/grass/branches/releasebranch_7_2/vector/v.generalize/simplification.c [Accessed 12 Jan 2018].
- [17] Jenks, G.F., 1981. LINES, COMPUTERS, AND HUMAN FRAILTIES. *Annals of the Association of American Geographers*, 71(1), pp.1-10.