Snap rounding in a triangulation

Fengyan Zhang

student #5462150

1st supervisor: Ken Arroyo Ohori
2nd supervisor: Hugo Ledoux

January 16, 2023

1 Introduction

Geometric computation is an essential aspect of geographic information science, which involves the use of mathematical algorithms to perform precise calculations on geometric shapes and spatial data. Complex polygons, which are often used to calculate areas and perform Boolean operations, can present issues when converting or exchanging datasets due to different representations used by various formats. These issues are often caused by limitations in precision, such as the use of float 32-bits in ESRI shapefile format (see `https://en.wikipedia.org/wiki/Shapefile`), resulting in vertices that are close to each other or close to other lines collapsing and creating invalid polygons, intersecting lines, and other errors that can prevent the use of these geometries in downstream applications. For instance, a self-intersected polygon (Fig. 1a) caused by coordinates shifting, a polygon that is not completely closed (the exterior ring is not closed, see Fig. 1b) will lead to 0 area. They are invalid polygons, which need to be carefully handled in the practice. It is important that the results of geometric computations are as accurate as possible so that the decisions made based on those results are reliable.

Obtaining reliable geometric computation results can be achieved by utilizing infinite precision techniques. This can be done through the use of libraries that support arbitrary-precision arithmetic (such as CGAL), or algorithms. These techniques allow for mathematical operations to be performed with an arbitrarily high level of precision, effectively eliminating potential errors caused by limitations in precision. It is important to note, however, that infinite precision may result in increased memory usage and slower performance. Additionally, it cannot guarantee precision when saving data to a file or converting files of different formats.

(a) A normal polygon (left) and a self-intersected polygon (right)

(b) Polygon with 0 area: a polygon that is not completely closed

Figure 1: Example: invalid polygons

Using appropriate algorithms to generate objects with finite-precision estimated coordinates is a common way to ensure the reliability of the results. Snap rounding (SR) is such a method to convert infinite-precision to finite-precision. SR snaps each point of the polygon onto an grid point within a prescribed tolerance, allowing for accuracy and adjustability depending on the needs of the application. Fig. 2 illustrates an example. SR works by "snapping" the vertices of a shape to a grid of equally spaced points, which can help to create well-separated vertices and cleaner geometric arrangements. By ensuring that the shortest distance between a vertex and its non-incident edge is a certain threshold, SR can help to organize shapes in a more structured way. It should however be noted that, standard SR will change the original topology to a greater or lesser extent, in other words, maintaining the topological shape is not its purpose. On the other hand, it cannot guarantee the correctness of the result. Iterated snap rounding (Halperin and Packer (2002)), which is widely used at present, is a good implementation for snap rounding algorithm. However it can be computationally expensive and slow in practice, particularly for large datasets, as it may require a significant amount of time and

space resources. To address these limitations, this thesis aims to improve the efficiency of snap rounding by using constrained Delaunay triangulation as a supporting data structure.

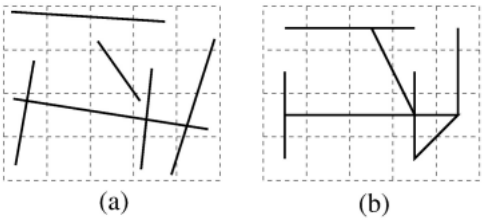

Figure 2: A snapping example from Halperin and Packer (2002): line segments before (a) and after (b) snap rounding

2 Related work

Halperin (2002) discusses the challenge of transforming geometric algorithms into effective computer programs. The paper starts by discussing the gap between theory and practice of geometric algorithms, and the difficulties caused by the basic assumptions of most theoretical geometric algorithms concerning complexity measures and robustness issues, specifically issues related to arithmetic precision and degenerate input. The paper then provides an overview of the CGAL project and library, which is a joint effort by several research groups in Europe and Israel to produce a robust software library of geometric algorithms and data structures. The library is now available for use with significant functionality. The author describes the main goals and results of the project. The central part of the paper focuses on arrangements (i.e., space subdivisions induced by geometric objects) and motion planning. The author concentrates on the maps and arrangements part of the CGAL library and describes two packages developed on top of CGAL for constructing robust geometric primitives for motion algorithms. The author also presents the CGAL project and library as a solution to these problems and describes how it can be used for arrangements and motion planning.

Goodrich et al. (1997) introduce snap rounding as a method for approximating geometric shapes defined using floating point coordinates with a discrete grid of points. The method is based on the idea of "snapping" the vertices of the shape to the nearest grid point, and then connecting the snapped vertices with straight line segments to form an approximation of the original shape. One downside of their methods is that they take longer to run as the number of input line segments and segments in a snap-rounded representation increase.

Although snap rounding is commonly used to improve the efficiency of algorithms that operate on geometric data, standard snap rounding algorithms can yield artifacts and degeneracies under some circumstances, such as intersections that should not exist or lines that are not properly connected. To address these limitations, de Berg et al. (2007) proposed a new intersection-sensitive snap rounding algorithm that approximates real-valued coordinates while preserving the topological features of the input geometry, such as intersections and adjacencies. This method was expected to produce fewer artifacts and degeneracies in some cases compared with the standard snap rounding method.

Hershberger (2013) introduces the concept of stable snap rounding, which is a variant of the standard snap rounding. It guarantees the stability of the produced approximations under perturbations of the input data while also preserving all of snap rounding's advantages and being idempotent. This is achieved by using a new criterion which ensures that the approximations are not sensitive to small changes in the input. The author also provides a detailed analysis of the stability properties of the algorithm through a series of experiments.

The robust geometric computing can be more or less achieved by the standard snap rounding. However, it is worth pointing out that in a snap-rounded arrangement, the distance between a vertex and its non-incident edge can be extremely small compared with the width of a pixel in the tiling grid (Fig. 3 shows an example). In order to solve this, Halperin and Packer (2002) proposed an enhanced algorithm - Iterated snap rounding (also know as ISR). ISR ensures that the distance between a vertex and its non-incident edge is at least half-the-width-of-a-pixel, the arrangement after rounded can be further improved in this way. The authors provide a detailed description of the algorithm, as well as a proof of its correctness. They also present experimental results comparing iterated snap rounding to the standard snap rounding algorithm. A conclusion has been drawn that the iterated snap rounding outperforms the original snap rounding algorithm in terms of the quality of the approximations. It should be noted that, the run time of ISR is relatively long, because the snap rounding operation needs to be performed many times. Although the authors used kd-tree as an auxiliary data structure to optimize, it is still computational intensive. For instance, for a map of the USA (containing about 56 polygons and 486 segments intersecting only at endpoints), the total average running time is about 78.64 seconds (from Halperin and Packer (2002)).

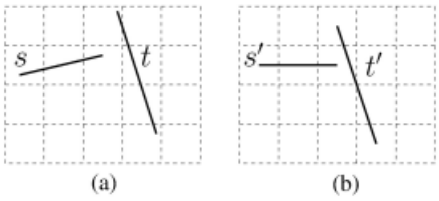

Figure 3: An example from Halperin and Packer (2002): a vertex is very close to a non-incident edge after (b) snap rounding

Packer (2006) present a new variant of the Snap Rounding and Iterated Snap Rounding method, called Iterated Snap Rounding with Bounded Drift (ISRBD). The authors note that both Snap Rounding (SR) and Iterated Snap Rounding (ISR) have limitations, such as the potential for degeneracies in SR and drift in ISR. The ISRBD method proposed in the paper aims to overcome these shortcomings by augmenting the ISR method with procedures that guarantee the quality of the geometric approximation of the original segments, while still maintaining the property that a vertex and a non-incident edge in the rounded arrangement are well separated. The authors investigate the properties of the new method and compare it with the earlier variants. They also implemented the new scheme on top of the Computational Geometry Algorithms Library (CGAL) and report on experimental results.

Belussi et al. (2016) introduce a new algorithm called Snap Rounding with Restore (SRR) that aims to improve the robustness and quality of geometric approximation in datasets while preserving the topological structure. The algorithm is an improvement on the well-known Snap Rounding algorithm, by eliminating configurations in which the distance between a vertex and a nonincident edge is smaller than half the width of a pixel of the rounding grid. The goal of SRR is the same as another algorithm called Iterated Snap Rounding (ISR) and its evolution, Iterated Snap Rounding with Bounded Drift (ISRBD). However, the paper claims that SRR produces an output with a better quality of approximation, both in terms of the distance from the original segments and the conservation of their topological structure. This is supported by a statistical analysis on a large collection of input datasets. The paper also notes that ISRBD may produce strong topological modifications, which SRR does not. The conclusion is that SRR is suitable for applications that require robustness, guaranteed geometric approximation, and good topological approximation.

Accurately representing geographic networks at reduced coordinate precision can be chal-

lenging. Specifically, it requires that vertices need to be placed on a grid and that the network topology is maintained, without the introduction of intersections or collapse of line segments or faces. The problem of minimizing the "rounding error" is known to be NP-hard and practical methods are difficult to implement. van Dijk and Löffler (2019) propose a two-stage simulated annealing algorithm which first focuses on finding a feasible solution, and subsequently optimizes the rounding error. Furthermore, the authors also examine various feasibility procedures and evaluate their applicability on geographic networks. The dataset and an implementation in C++ can be accessed at `https://github.com/tcvdijk/armstrong`.

Related to the snap rounding, 3D geoinformation research group at TU Delft has previously used a constrained triangulation as a robust method to repair polygons (Ledoux et al. (2014)) and planar partitions (Ohori et al. (2012)). Topological errors can be automatically fixed in such a manner. Utilizing constrained triangulation as a supporting data structure appears to be a logical and efficient method for optimizing the snap rounding process. For example, it eliminates the need for constructing a tiling grid for the entire area and allows for points to be placed anywhere, rather than solely on a grid. This can enhance both the speed and memory efficiency.

3 Research questions

The main research question of this thesis would be: *How can a constrained Delaunay triangulation be used as a supporting data structure to efficiently perform snap rounding?*

This thesis will be focusing on investigating and optimizing the use of constrained Delaunay triangulation for performing snap rounding, through the development and implementation of efficient algorithms and the evaluation of their performance in various scenarios. In order to achieve this goal, the following sub-tasks will be gradually studied:

- To develop and implement an efficient algorithm for performing constrained triangulation based snap rounding on a given set of points and polygons.
- To investigate the accuracy and reliability of constrained triangulation based snap rounding compared to other methods of approximation (e.g. the iterated snap rounding).
- To evaluate the applicability of constrained triangulation based snap rounding in various real-world scenarios and domains, such as computer graphics, geo-spatial analysis, and scientific visualization.
- Is it better to have Delaunay triangulation or is any type of constrained triangulation fine?
- How to handle the reassignment of triangles when they are modified by the snapping process? For example, If the triangulation was originally derived from polygons, we need to determine which polygon each new triangle should belong to after the snapping process.
- In the GIS field, there are not only line segments, but also polygons with attributes or semantics. Whether they can be preserved or how to preserve them in a proper way during the snapping process needs to be studied.
- Try to evaluate the performance of the implemented method (e.g. run time & space resources occupied at runtime).

This thesis will examine the use of constrained (Delaunay) triangulation for snap rounding and focus on the process of valid polygons. Handling invalid polygons (such as self-intersecting, etc.) is not within the scope of discussion. For more details, please refer to Ledoux et al. (2014) and Ohori et al. (2012).

4 Methodology

The overall idea is to establish a constrained Delaunay triangulation for the input geometry, taking the edges of the input geometry as constraints. By identifying the points/lines/faces that need to be snapped under different circumstances (which will be discussed in more detail later), the overall data can be snapped and the revised geometry can be output, while maintaining the integrity of the constraints as much as possible.

A constrained triangulation is a type of triangulation that ensures that certain edges, known as constraints, are present in the triangulation. This is useful in situations where the input geometry has certain features that must be preserved in the triangulated version(e.g. the constrained edges need to be maintained). The process of creating a constrained Delaunay triangulation involves reading in the data, establishing constraints based on the geometric characteristics of the input data, and then creating the corresponding triangulation. It is important to pay attention to issues such as edge crossing and overlap when establishing constraints.

The snapping process involves identifying the points, line segments, and faces in the input geometry that need to be snapped, or adjusted, to fit the constraints. This can involve moving or adding points to ensure that the constraints are satisfied. Once the data has been snapped to fit the constraints, the revised geometry is output, with the goal of preserving the original constraints as much as possible.

Fig. 4 displays a theoretical flowchart of the main process of this thesis.

4.1 Preprocessing

Preprocessing is an important step in the snap rounding process, as it helps to ensure that the data is in a suitable status for further processing. To be more specific, snapping process should preferably be based on valid polygons. While in most cases, original data sets are usually chaotic and may contain errors or inconsistencies that can hinder downstream analysis. In the case of geometric data, such as polygons, tools such as prepair (Ledoux et al. (2014)) and pprepair (Ohori et al. (2012)) may be used. These tools help to identify and fix errors or inconsistencies in the polygons, ensuring the validity of the geometric shape.

4.2 Snapping

4.2.1 Snapping case 1: trivial faces

In this step, our primary goal is to identify and snap trivial faces, which we will refer to as "snapping triangles." These are represented by triangles with all side lengths less than or equal to a predetermined snapping threshold.

A preliminary result of snapping case 1 is shown in Fig. 5. The detailed steps are as follows.

- To determine the triangle with the smallest tolerance, each face handle is traversed and the lengths of the edges are obtained.
- The lengths of the edges are compared to a manually-specified tolerance. Only triangles with all three edges shorter than the threshold are considered as suitable for snapping.

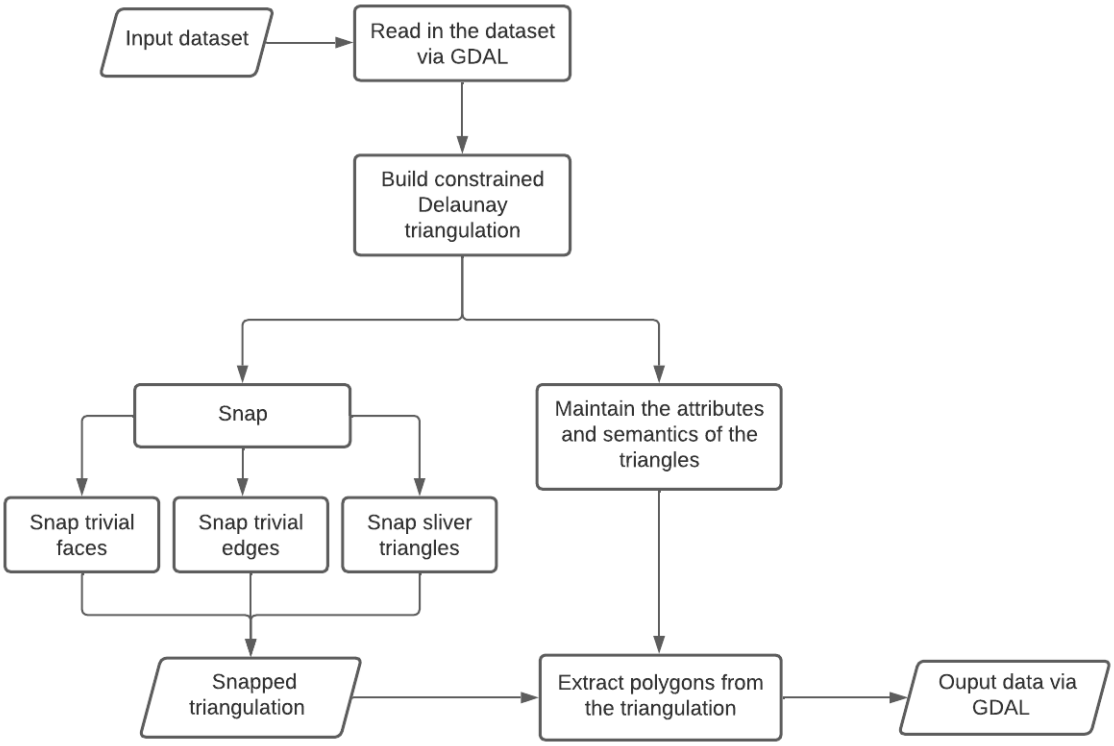

Figure 4: Flowchart of the main process

- All constraints incident to the selected triangle are identified by finding the constrained incident edges incident to each vertex of the triangle.
- The incident vertices are identified according to the incident constraints, those belonging to the triangle itself are filtered.
- Store the incident vertices.
- Remove the triangle by deleting its three vertices.
- Replace the triangle by its calculated centroid.
- Reintroduced the constraints between the calculated centroid and the incident vertices.

4.2.2 Snapping case 2: trivial edges

The main objective of this step is to identify and snap trivial edges, which we will refer to as "snapping edges." These are represented by edges with a length less than or equal to a pre-set threshold value.

A preliminary result of snapping case 2 is shown in Fig. 6. The detailed steps are as follows.

- To determine the edge with the smallest tolerance, the length of each edge is traversed and the lengths of the edges are obtained.
- The lengths of the edges are compared to a manually-specified tolerance. Only edges with edge length shorter than the threshold are considered as suitable for snapping.

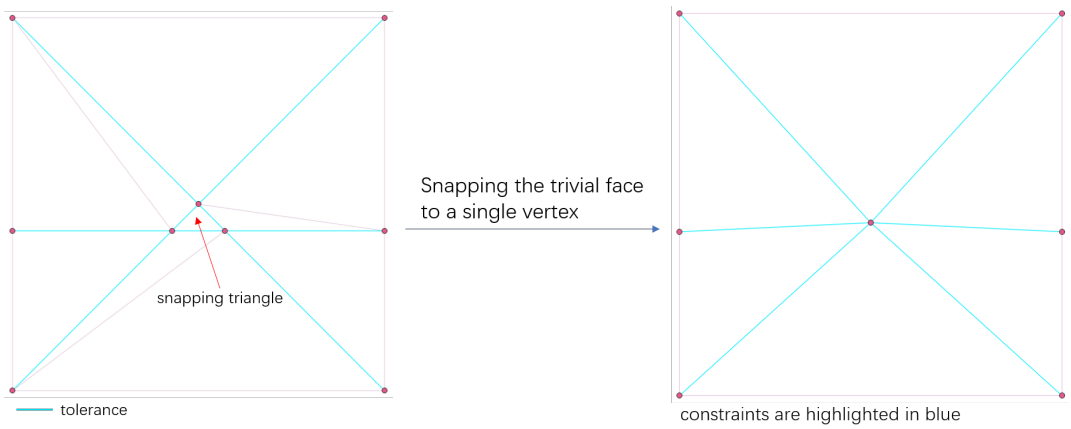

Figure 5: Case 1

- All constraints incident to the selected edge (snapping edge) are identified by finding the constrained incident edges incident to each vertex of the snapping edge.
- The incident vertices are identified according to the incident constraints, those belonging to the snapping edge itself are filtered.
- Store the incident vertices.
- Remove the snapping edge by deleting its two vertices.
- Replace the snapping edge by its calculated centroid.
- Reintroduced the constraints between the calculated centroid and the incident vertices.

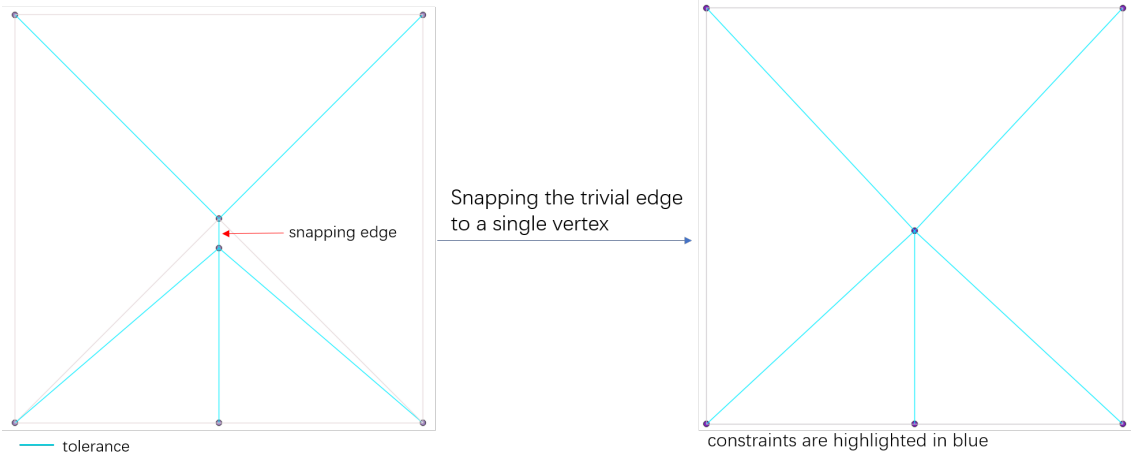

Figure 6: Case 2

4.2.3 Snapping case 3: sliver triangles

In the constructed triangulation, in addition to the triangles whose three sides are smaller than the threshold, there is another kind of triangle that we need to consider, that is, a triangle whose height is smaller than the threshold (a very sliver triangle). Here we consider the case where the basic edge of the height is a constrained edge. For such sliver shapes, there are two

possible snapping schemes: 1 For the base edge whose height is smaller than the threshold, mark it as unconstrained, and ensure that the other two edges are constrained (if they are not, mark them as constrained) 2. Keep only the base edge, and move the vertex opposite to the base edge to the base edge. The detailed steps can be described as follows.

- Traverse each face handle, find triangles with altitudes smaller than the tolerance and identify the constrained base edge.
- Remove the constrained base edge, keep the incident vertices unchanged.
- Ensure the other two non-base edges are now constrained, if note mark them constrained.
- Move middle vertex to lie on the removed constraint (the original base edge).

Fig. 7 shows an example.

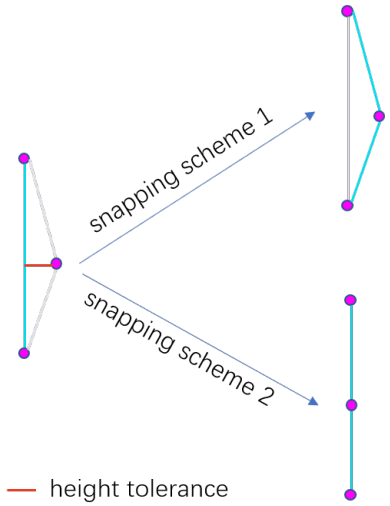

Figure 7: Case 3

The snap process should be able to handle not only polygons, but also line segments, or a combination of both. This should also be tested in subsequent practice.

4.3 Labelling triangles

Identifying the polygons from the triangulation during the snap process is of great importance as well. To make this process easier, we label all triangles in the triangulation as either outside or inside. We then retain or ignore the triangles based on the marks on the left and right sides of each edge in the triangulation. This will help us identify which triangles belong to which polygons, maintain their respective properties, and extract polygons from the triangulation. An illustration is shown in Fig. 8. More details can be found in Ohori et al. (2012).

Labeling triangles can be a difficult task when it comes to implementation, due to two main reasons. Firstly, the snap process can alter the local structure of the triangulation, making it hard to determine which polygon a new triangle should belong to. Secondly, when using Delaunay triangulation as a supporting data structure, it is necessary to constantly update the triangulation to preserve its Delaunay characteristics. This can lead to changes in the overall shape of the triangulation, rendering the initial labels of triangles meaningless. As a result, correctly labeling triangles in these cases poses a significant challenge.

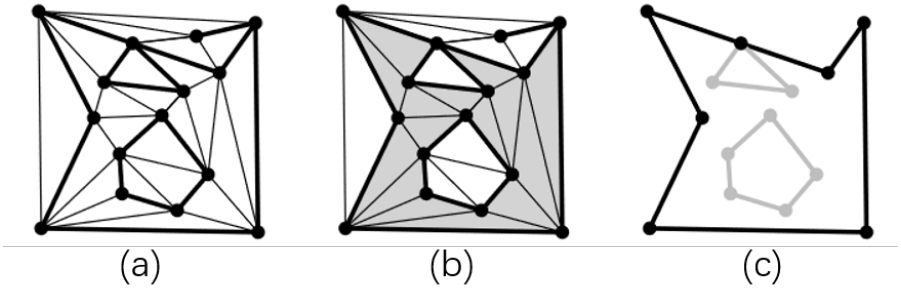

Figure 8: Labelling triangles example from prepair (Ledoux et al. (2014)). (a) Construct the triangulation; (b) Triangles are labelled as inside (grey) or outside (white); (c) Reconstructed polygons.

4.4 Assessment

Assessing the results of constrained-Delaunay-triangulation based snap rounding can be done by comparing it to the iterated snap rounding algorithm implemented in CGAL. One way to do this is by comparing the topological and geometric consistency of the snapped results obtained from both methods. We can also compare the running time of two methods by using the same input, and record the space resources used by each method as well. For example, Halperin and Packer (2002) compare the result of SR and ISR based on the same dataset (see Fig. 9). The results of this thesis can also be compared in the similar way.

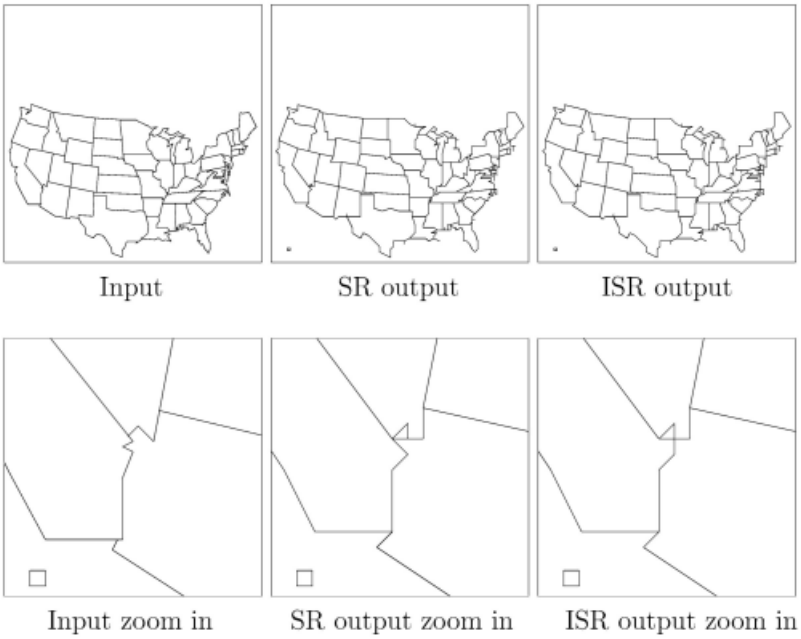

Figure 9: Comparison of SR and ISR output from Halperin and Packer (2002)

5 Time planning

The general graduation calender is shown below. The specific date will be determined according to the arrangement of the tutors and the college.

Event	Expected Date
P1: Registration of topics/mentors	December, 2022
P2: Graduation plan (formal assessment)	January(end), 2023
P3: Midterm progress meeting	March(end), 2023
Submit draft thesis	May(mid), 2023
P4: Go/no-go (formal assessment)	May(end), 2023
Submit final thesis	June(mid), 2023
P5: Public presentation and final assessment (formal assessment)	June(end), 2023

The Gantt chart (displayed in Fig. 10), provides a clearer overview of the estimated timeline for major events and tasks. It is important to note that this is a simplified diagram and more specific tasks will be included in the follow-up process.

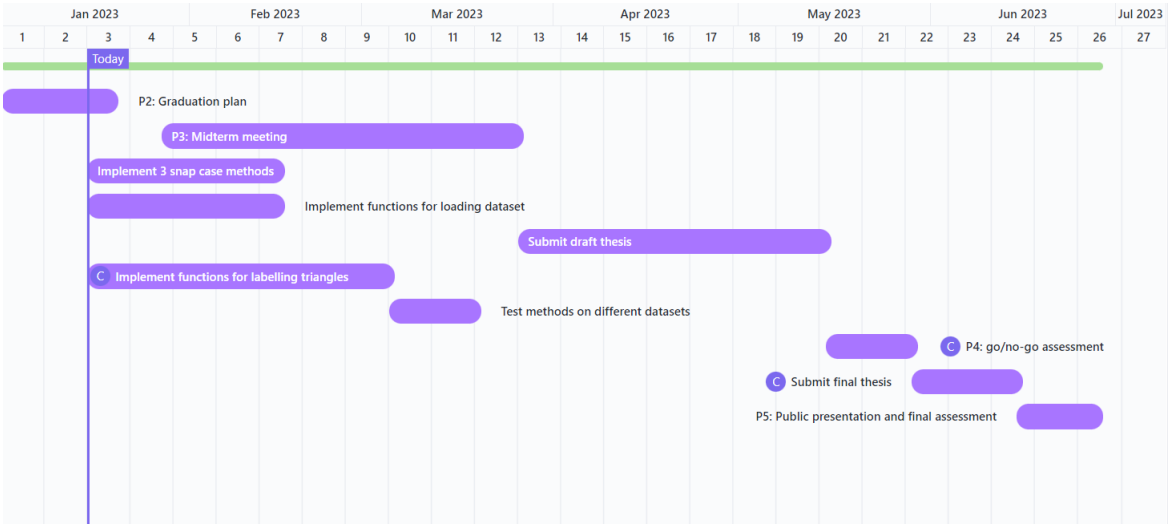

Figure 10: Overview of events and tasks

6 Tools and datasets used

6.1 Tools

This thesis will mainly use C++ as the main programming language, use the Computational Geometry Algorithms Library (CGAL) and the Geospatial Data Abstraction Library (GDAL) to implement the constrained-Delaunay-triangulation based snap rounding method. CGAL is an open-source library for computational geometry that provides a wide range of geometric algorithms, including Delaunay triangulation. GDAL is an open-source library for reading and writing geospatial data, which will be used to read and write datasets in various formats, for example GPKG and ESRI shapefile.

Additionally, we will be using two prototypes, prepair(available at: `https://github.com/tudelft3d/prepair`, details can be found in Ledoux et al. (2014)) and pprepair (available at: `https://github.com/tudelft3d/pprepair`, details can be found in Ohori et al. (2012)), to ensure the overall quality and validity of GIS polygons. The prepair prototype allows for easy repair of "broken" polygons according to the international standard ISO19107. Given a single input polygon, it will automatically repair it and return a valid polygon. The pprepair prototype takes a set of polygons and ensures that they form a valid planar partition, with no gaps or overlaps. It can identify problems in individual polygons or in the overall planar

partition, and also includes an automatic repair feature to output a set of polygons that do form a valid planar partition.

6.2 Datasets

In terms of datasets, this thesis will use OpenStreetMap (also known as OSM, available at: https://www.openstreetmap.org/) to obtain various datasets. OSM is a collaborative project that provides free, editable maps of the world. This thesis will also use various data formats, for example GeoPackage (GPKG) and ESRI shapefile, to evaluate the performance of the proposed method. The datasets usually include a variety of geographic features, such as roads, buildings, and bodies of water, yet we will mainly focus on the 2D and 2.5D polygons. The datasets will be chosen to represent a range of different geographic regions and scales.

References

A. Belussi, S. Migliorini, M. Negri, and G. Pelagatti. Snap rounding with restore: An algorithm for producing robust geometric datasets. *ACM Trans. Spatial Algorithms Syst.*, 2(1), mar 2016. ISSN 2374-0353. doi: 10.1145/2811256. URL https://doi-org.tudelft.idm.oclc.org/10.1145/2811256.

M. de Berg, D. Halperin, and M. Overmars. An intersection-sensitive algorithm for snap rounding. *Computational Geometry*, 36(3):159–165, 2007. ISSN 0925-7721. doi: 10.1016/j.comgeo.2006.03.002. URL https://www.sciencedirect.com/science/article/pii/S0925772106000356.

M. T. Goodrich, L. J. Guibas, J. Hershberger, and P. J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. pages 284–293, 1997.

D. Halperin. Robust geometric computing in motion. *The International Journal of Robotics Research*, 21(3):219–232, 2002. doi: 10.1177/027836402320556412. URL https://doi.org/10.1177/027836402320556412.

D. Halperin and E. Packer. Iterated snap rounding. *Computational Geometry*, 23:209–225, Sept. 2002. doi: 10.1016/S0925-7721(01)00064-5.

J. Hershberger. Stable snap rounding. *Computational Geometry*, 46(4):403–416, 2013. ISSN 0925-7721. doi: 10.1016/j.comgeo.2012.02.011. URL https://www.sciencedirect.com/science/article/pii/S0925772112001551.

H. Ledoux, K. Arroyo Ohori, and M. Meijers. A triangulation-based approach to automatically repair gis polygons. *Computers & Geosciences*, 66:121–131, 2014. ISSN 0098-3004. doi: 10.1016/j.cageo.2014.01.009. URL https://www.sciencedirect.com/science/article/pii/S009830041400020X.

K. A. Ohori, H. Ledoux, and M. Meijers. Validation and automatic repair of planar partitions using a constrained triangulation. *Photogrammetrie - Fernerkundung - Geoinformation*, 2012(5):613–630, 10 2012. doi: 10.1127/1432-8364/2012/0143. URL http://dx.doi.org/10.1127/1432-8364/2012/0143.

E. Packer. Iterated snap rounding with bounded drift. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, page 367–376, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933409. doi: 10.1145/1137856.1137910. URL https://doi-org.tudelft.idm.oclc.org/10.1145/1137856.1137910.

T. C. van Dijk and A. Löffler. Practical topologically safe rounding of geographic networks. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '19, page 239–248, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369091. doi: 10.1145/3347146.3359347. URL https://doi-org.tudelft.idm.oclc.org/10.1145/3347146.3359347.