Implementation of the QuickOpp Algorithm

As defined in “Complete Coverage Path Planning in an Agricultural Environment”

By

Andrew Stelter

A report submitted to Dr. L. Pyeatt

in partial fulfillment of the requirements for the degree of

Master of Computer Science and Robotics

South Dakota School of Mines and Technology

Rapid City, South Dakota

Semester Completed: Summer 2020

Acknowledgements

My parents, who have supported me through my entire college education and pushed me to strive for excellence.

Stephen Fjelstadt, who inspired this project and who has provided moral support when the Real World gets busy.

Dr. Larry Pyeatt, who, despite my continual delays, helped guide me through the murky waters of understanding another's research project.

Table of Contents

List Of Images

Introduction

One of the biggest technological advances of our time has been the development of artificial intelligence and smart systems which can automate tasks previously possible only for humans. While there are many applicable fields for this technology, one of the biggest is the operation of vehicles such as cars, trucks, or farm equipment.

In specifically the agricultural field, there is a significant interest in algorithms pertaining to “Complete Coverage Path Planning” (or CCPP). CCPP is the action of planning a route that allows a vehicle with some width associated with it to cover an entire area. For example consider a roomba - ideally, your robot vacuum will travel a path that allows it to vacuum your entire carpet and do so as quickly as possible so as to minimize disruption to your daily life; CCPP is the type of algorithm used to determine the path the roomba should follow. Such a concept is magnified in importance when scaled up to be applied to a farm; if you plant your field in an efficient manner, it can cut out miles of driving over the course of a season as you water and spray your crops multiple times and then harvest them. Miles not traveled translates directly to fuel saved which in turn translates to lower operational costs for the farmers involved, so if a method could be found to determine the best possible path to cover a field, it could be used to ensure the maximum amount of savings.

Unfortunately, Complete Coverage Path Planning can be reduced to a variation on the well-known Travelling Salesman Problem and therefore is an NP-complete problem, so finding such an ideal solution is currently not reasonable. Meuth (2001), Zelinsky (1993). This remains a relevant topic of research, and numerous attempts have been made to find solutions for producing path plans, some including stochastic or evolutionary approaches to the problem. Khan (2017).

A common theme amongst algorithms which do not take a probabilistic approach is the decomposition of the original area into pieces which can be planned more easily as separate problems and then recombined into a final solution. In 2009, Dr. Jian Jin designed such an algorithm specifically in the context of farming which produces excellent results in O(n^3 lg n) time (in the number of edges on the input shape). Jin's work is applicable in both two-dimensional and three-dimensional cases, taking into account hills and valleys in the field. A subset of this work was used as the basis for the QuickOpp algorithm described in the paper "Complete Coverage Path Planning in an Agricultural Environment" by Theresa Driscoll in her Master's work in 2011. Driscoll's algorithm improves the runtime of the algorithm to O(n^2), drastically reducing runtime for large shapes by using a heuristic to pick only one direction to use when subdividing the field.

This project is an implementation of the QuickOpp algorithm, designed with common software engineering principles kept in mind and written using modern C++ features and libraries. The implementation includes the following deliverables

- A library implementing the QuickOpp algorithm built using the Boost library (specifically Boost.Geometry)
- A desktop application which allows users to load field data in the geojson format, process it using QuickOpp, and display the results
- A suite of tests for testing well-defined pieces of the algorithm with objectively correct solutions, such as the trapezoidal decomposition and cost function.

Terms

In the context of this report, please keep the following definitions in mind

- Point - A 2d geometric location
- Ring/Loop - A sequence of points, with an implicit connection between the first and last such that they form a closed area.
- Polygon/Shape - A collection of rings, with the first one defining some outermost area and all other rings being contained fully within the outer ring defining holes. Given these rings, the geometric difference of the outer area and all inner areas gives the target area or the 'inner' area. Conventionally, the outer right is defined in clockwise order and the inner rings are defined counter-clockwise.
- Field - A polygon used to indicate that the inner area is the target to be covered in an agricultural context, whether this means it should be planted, sprayed, or have some other operation applied.

- Machine - A drivable piece of machinery with some width associated with it such that driving in a straight line produces a rectangular area that is ‘covered’ by the machine.
- Swath - Single pass driving across a field in a machine
- Region - A ring which contains only inner area from a shape. These are created via the decomposition algorithm.
- Left/Right/Top/Bottom - Directional terms are used to reference relative pieces of the shapes in this algorithm. ‘Left’ denotes having a smaller X coordinate value, and ‘Right’ denotes a larger one. Similarly ‘bottom’ or ‘below’ and ‘top’ or ‘above’ denote having smaller Y and larger Y coordinates respectively.

Background

Without restating too much of the original paper, this section will describe the QuickOpp algorithm and explain how it compares to the original work by Jin. Like Jin's work, QuickOpp is designed for agricultural applications. This adds some restrictions that may not be found in more generalized CCPP research; one restriction is that farm vehicles are usually designed using a standard Ackermann steering system and are therefore limited in how they can turn. A roomba is able to turn in place, but most farm vehicles drive like cars and must travel forward while steering with the front wheels. While there are exceptions, such as tread-driven tractors, front-steered tractors are common enough that it is desirable for the algorithm to accommodate them.

There are some important distinctions between the works of Driscoll and Jin. Jin's algorithm was designed to take into account fully three-dimensional data - QuickOpp operates only in a two-dimensional plane. More importantly, QuickOpp does not perform the exhaustive search of the problem space that Jin's algorithm does; Jin's algorithm attempts to find the best possible method of subdividing a region and covering it, and this is what leads to the long runtime. QuickOpp makes the assertion that the best overall direction for the field is likely to be the best direction for most of the subregions and uses that direction to subdivide the field only one way; it then adds a step to recombine adjacent subdivisions similarly to the algorithm designed by Oksanen (2007).

QuickOpp can be broadly described as having three steps:

- Determine a best way to cover the field the field with a single driving direction and calculate the cost of this approach
- Segment the field using a modified trapezoidal polygon decomposition based on the work of Oksanen (2007)
- Recombine adjacent segments of the field if the cost-savings of doing so produces a lower overall cost

Cost Functions and Initial Cost

A portion of Driscoll's paper is used to justify their choice of cost function, which matches the cost function chosen by Jin. Ultimately, the goal is to minimize the total distance travelled while covering the field; this is the sum distance of all passes across the field (sometimes called swaths) and turns used to change direction at the end of a pass; however, neither Dricoll nor Jin included the distance of driving across the field in their cost functions. As Driscoll shows, the cost of turning is significantly higher than the cost of traveling straight except for cases in which the width of the vehicle is large compared to the overall shape the turning cost dominates the overall cost. With this in mind, it is acceptable to use only the distance travelled while turning around to formulate a cost function.

Like Jin, Dricoll used the cost function for one specific type of turn, which is dubbed the U-Shaped turn. In a U-Shaped turn, the driver travels forward until they have left the field entirely, turns around by driving a half-circle, and then travels forward until they have entirely entered the field. This description provides three 'pieces' of the turn which are weighted separately, as they may be driven at different speeds and have different overall costs to the turn. Figure 1 depicts a turn of this nature.

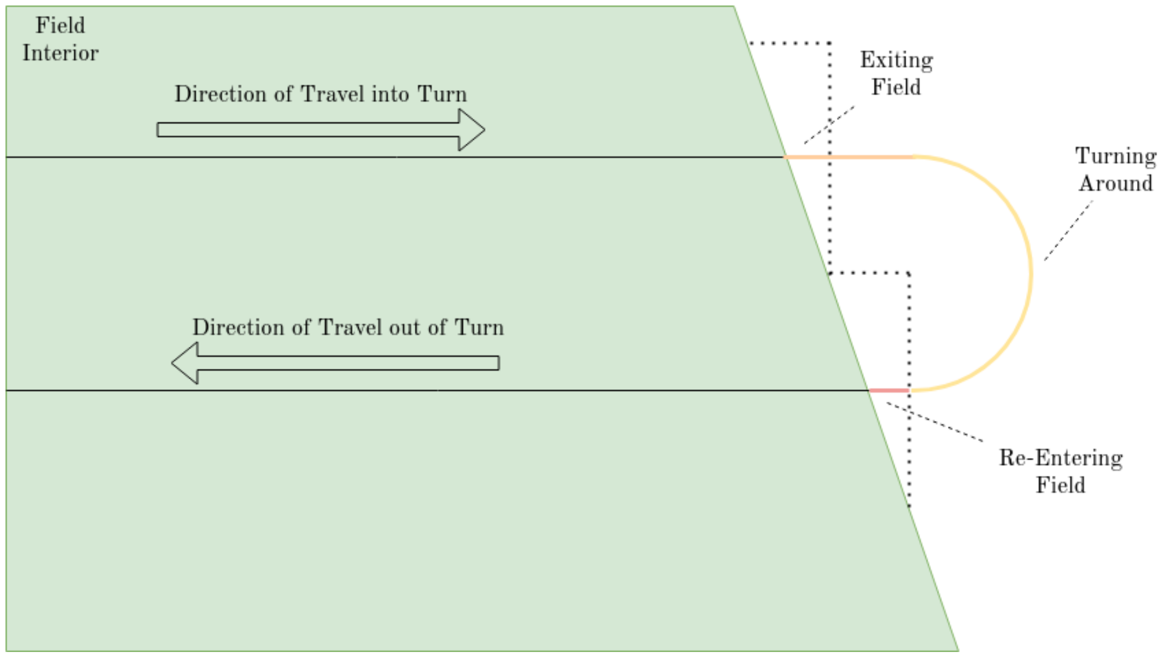

Field Interior
Direction of Travel into Turn
Exiting Field
Turning Around
Direction of Travel out of Turn
Re-Entering Field

Figure 1. The layout of a U-Shaped turn, consisting of three parts - exiting the field, turning around, and re-entering the field

The function used to compute the distance travelled in such a turn can be derived with basic geometric formulas. As shown by Driscoll, this formula is the following:

$$DST = 2w \;/\; |\tan(|\Theta - \beta|)| + \pi r$$

Where Θ is the angle of travel, β is the angle of the edge of the field, w is the width of the machine, and r is the radius of the turn (which is equal to $w/2$). Given an edge i on the field, the cost of all turns on that edge is equal to the cost of turning around on that edge multiplied by the number of turns (which may be a fraction). The number of turns taken on an individual edge with length L_i and angle β_i is also shown by Driscoll. It is

$$N_i = L_i \sin(|\Theta - \beta_i|) \ / \ 2w$$

Using these two equations together, Driscoll shows that the total cost of an edge is then the following:

$$\begin{aligned} COST_i &= DST_i{*}N_i \\ &= (L_i * |\cos|\Theta - \beta_i|)| \ / \ 2) + (L_i * |\cos(|\Theta - \beta_i|)| \ / \ 2) \\ &\quad + (\pi \, L_i * \sin(|\Theta - \beta_i|) \ /4) \end{aligned}$$

This equation keeps the three separate pieces of the turn split so that it is easy to apply weight values to them later on in the paper. Curiously, this solution shows that the width of your vehicle is not actually relevant, and that the cost of an edge is constant given its angle and the angle of travel.

Given this cost function for a single edge of the shape, we can see that the total cost of all turns taken while covering the field in a specific direction will be the sum of $COST_i$ for all edges on the shape, including edges of inner rings.

Once a cost function is chosen, determining an initial cost of the shape is a matter of iterating over all directions in a range of 180 degrees and determining the cost of the shape for traveling at that angle. If the costs of these angles of travel formed a differentiable function, it would be possible to find the absolute minimum and use that; however Driscoll showed that this is not the case, so we must explicitly check each angle (with some defined delta) and pick the best one.

Polygon Decomposition

Once an initial direction and cost is established, that direction is used to pick the sweep direction for the decomposition step. The sweep line is chosen perpendicular to the best-case initial direction of travel. The logic behind this choice is that if there is a direction which works well for the field as a whole, it is likely that that direction is appropriate for most of the field, so we want to bias the decomposition algorithm toward producing output in that direction unless it is extremely costly to do so. Figure 2 shows a simple L-shaped field which has an obvious best-case direction and demonstrates how this relates to the decomposition sweep-line.

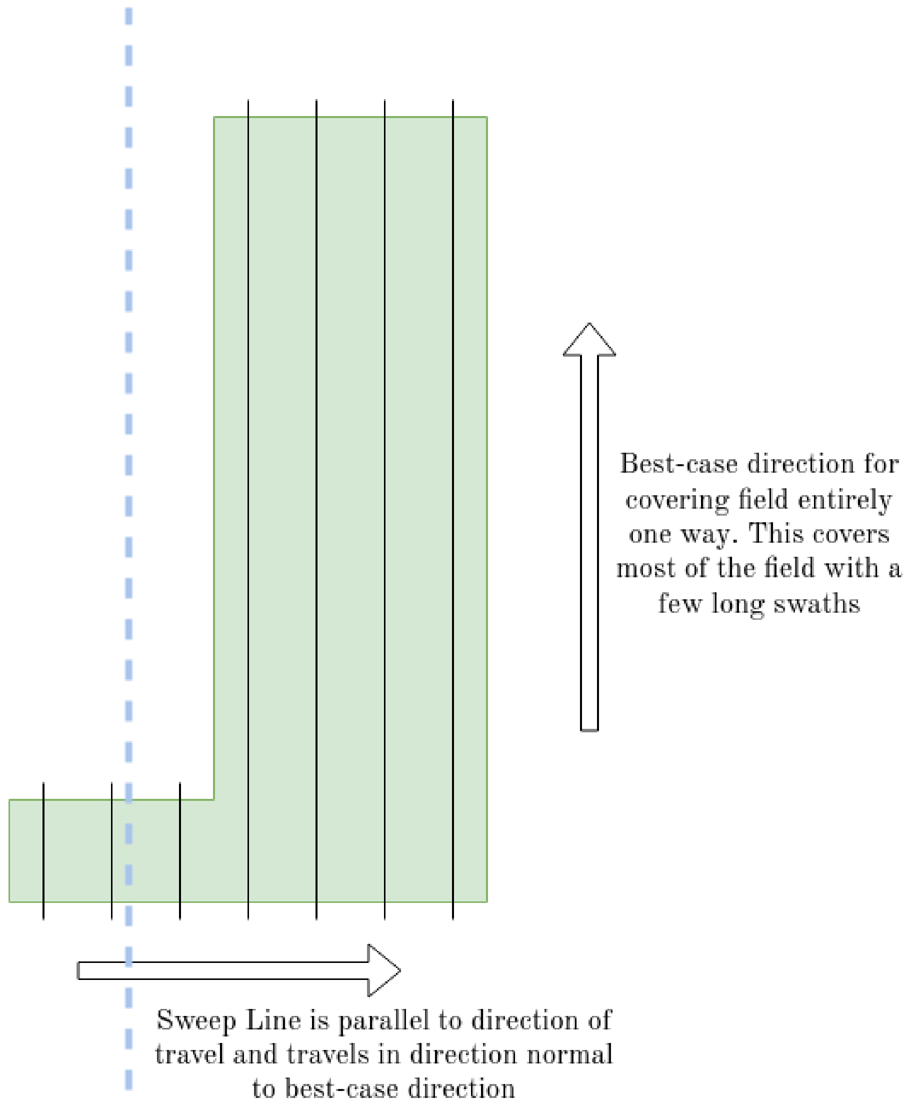

Best-case direction for covering field entirely one way. This covers most of the field with a few long swaths
Sweep Line is parallel to direction of travel and travels in direction normal to best-case direction

Figure 2. Example of choosing initial direction for an L-Shaped field. An initial direction is chosen that minimizes cost for the field as a whole, even if it is not ideal for portions of the shape.

The decomposition step of this algorithm takes as its input the shape defining the field in question and outputs a doubly-connected edge list defining the segments for the shape. Doubly-connected edge lists (also known as DCELs) are a common data structure for geometric algorithms which define shapes in terms of 'regions', 'half-edges', and 'vertices'. The half-edges form a double-linked list to form a ring which encloses a region, and the half-edges can have a 'twin' edge which uses the same points but travels the other direction and is part of a different ring around a different region.. Figure 3 shows an example of a DCEL.

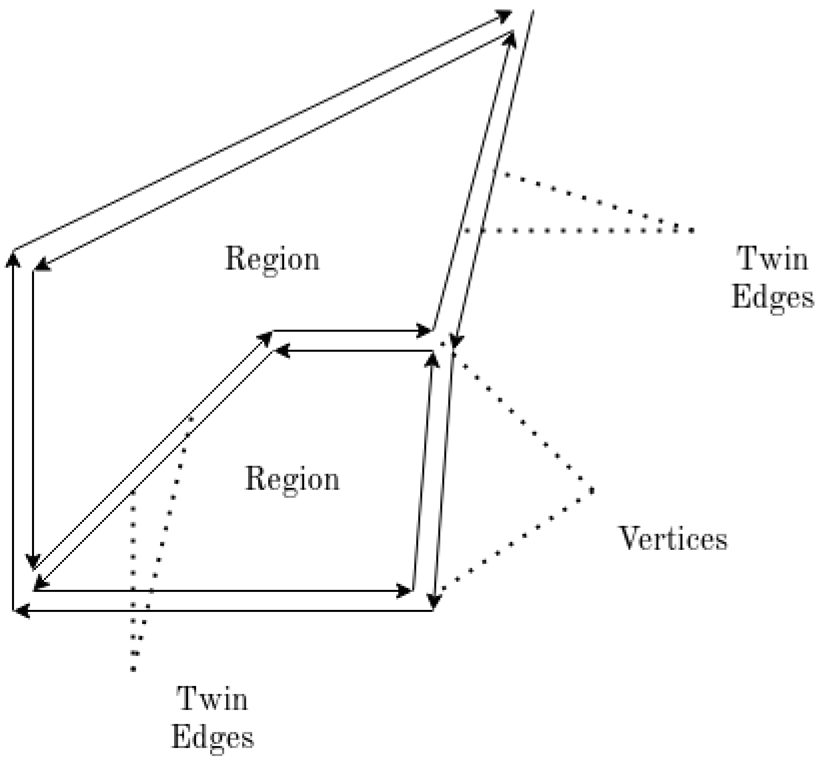


Figure 3. A simple DCEL with two regions, demonstrating how half-edges can share vertices and have twin edges which travel the opposite direction.

The sweep-line algorithm is defined in full in the QuickOpp paper; here is a brief summary

- Sort all edges of the input from left to right, and then from bottom to top.

- For each edge in this list
    - Add the edge to the active edges list
    - If it is the start of an outer loop of the shape
        - Add a new region to the DCEL, and add this edge to it
    - If it is the start of an inner loop of the shape
        - Create a vertical edge which splits the shape apart and creates two new regions, one above and one below the hole that was encountered
    - If it is the end of an inner loop of the shape
        - Mark the edge as unfinished
    - Otherwise if the sweep line has traveled far enough and there is a sharp angle between this edge and the next, create a vertical edge which splits the shape and create a new region on the right-hand side
- If the sweep-line has traveled past the end of any unfinished edge, create a vertical edge to split the shape there, closing the two regions above and below the inner hole and creating a new region to the right-hand side.

The process of decomposing a simple shape with a hole in the middle can be seen in figure 4. This is a simplification of the algorithm, but it shows how the sweep line travels across the shape and builds the DCEL as it goes.

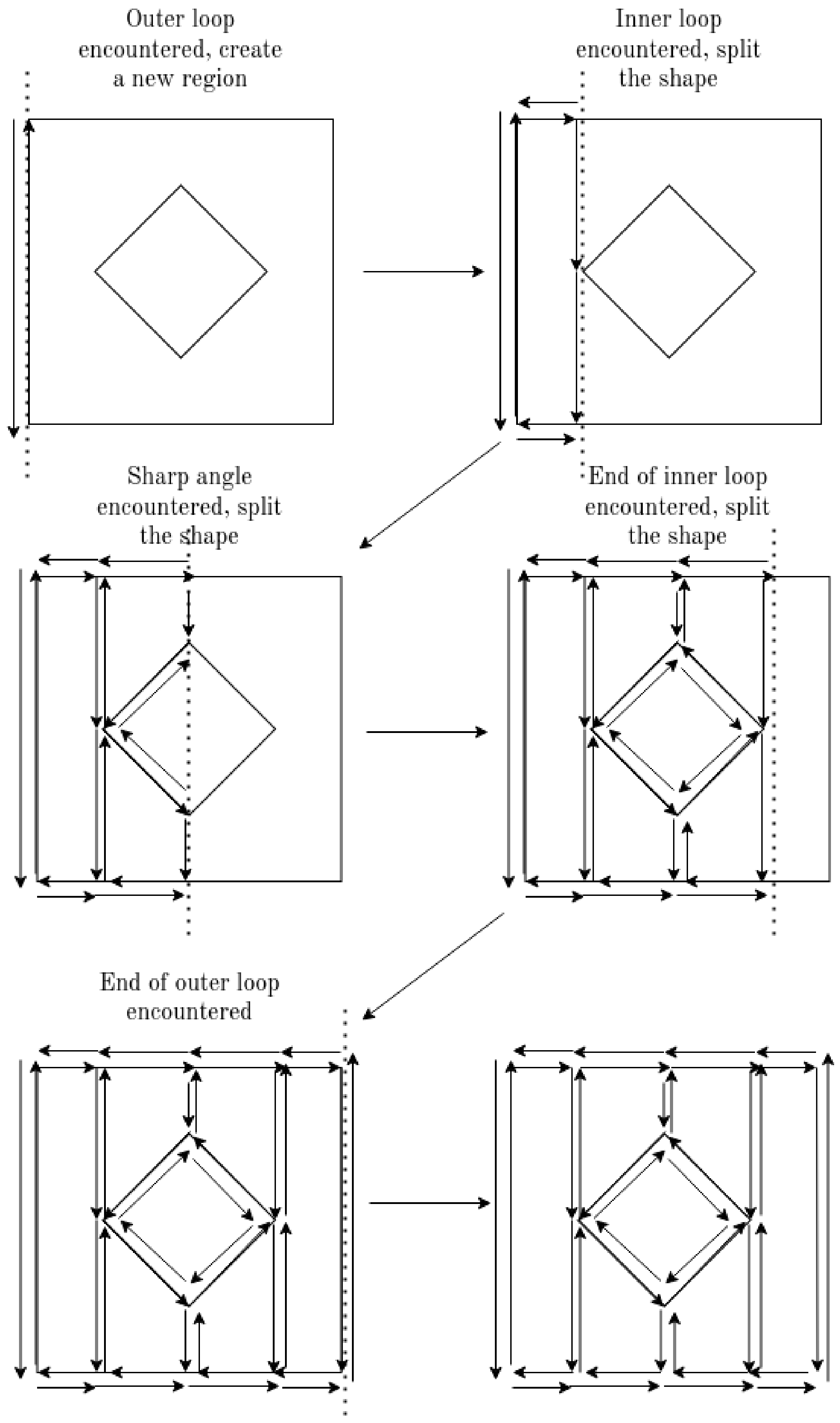

Outer loop encountered, create a new region
Inner loop encountered, split the shape
Sharp angle encountered, split the shape
End of inner loop encountered, split the shape
End of outer loop encountered

Figure 4. Demonstration of the polygon decomposition algorithm

This decomposition algorithm is similar to the algorithm used by Oksanen and Visala in their 2007 research. Unlike that algorithm, which used a standard trapezoidal decomposition (splitting the shape at every step of the algorithm), the decomposition in QuickOpp only splits at places where the edges of the form a sharp angle (‘sharp’ was defined as greater than 10 degrees). This produces a decomposition where each piece is approximately trapezoidal, and significantly reduces the number of regions which need to be merged in the later step.

Merging Regions

The last step of QuickOpp is to consider different methods of merging existing regions to produce the final result. As QuickOpp is defined, each region will have at most two adjacent regions on both the left and right-hand sides. In cases where there might logically be more adjacent regions (such as when there are two holes which end at exactly the same X coordinate) this invariant is maintained by creating infinitely thin regions which connect to only some of the nearby regions.

Before merging any regions, first the best-case cost is calculated for each individual region using the same method as was used to calculate a best overall cost at the beginning of the algorithm.

Then, the merging step traverses the regions in a depth-first ordering, treating adjacent regions as adjacent nodes in a graph. At each step, the algorithm considers the cost savings of merging the two active regions and using various methods of covering them. The options considered are the following

- Cover both with the best direction of the left
- Cover both with the best direction of the right
- Cover both with the normal direction to the best direction of the left
- Cover both with the normal direction to the best direction of the right
- Cover both with their original directions, creating a bend along the edge the share.

The QuickOpp paper explains in detail why these are the options considered and how to calculate the cost savings for each of them.

Project Design

This section will explain the design of the project, give some details about the implementation, and describe modifications made to the QuickOpp algorithm during development.

Project Design

As mentioned previously, this project is split into three parts

- Implementation of QuickOpp as a library
- A UI for executing QuickOpp
- Unit Tests

Development of a UI and unit tests along with the QuickOpp algorithm provided constant feedback as to the state of the project. This is valuable for verifying that the results made sense visually and that the internal algorithm pieces were behaving as expected.

QuickOpp Library

While the QuickOpp library is focused around the QuickOpp algorithm, two pieces of the algorithm have been separated completely. All logic for the DCEL has been moved into a Dcel class which manages the integrity of the structure, and the logic for traversing a polygon via a sweep line has been separated as well. The Quick Opp algorithm itself is the largest piece of the library.

The decision to break the DCEL structure and Sweep Line Algorithm into separate pieces was made after most of the project was developed. This was done for a few reasons

1. It keeps the logic of the QuickOpp algorithm focused on what is special about QuickOpp and hides the details of building a DCEL or performing a sweep algorithm.
2. DCELs and Sweep Lines are fairly common algorithms, and writing QuickOpp in terms of more generalized versions of them allows that generalized logic to be shared with other projects in the future.
3. Separating these pieces of the algorithm allows them to be unit-tested for correctness, to ensure the sweep line traverses points in the correct order and the DCEL produced is valid.

DCEL

The DCEL type encapsulates the logic needed to build and modify a doubly-connected edge list in ways that produce a consistent result. It hides all of the pointer management required to connect the various pieces of the DCEL, and provides access through handle objects which represent specific pieces of the DCEL. In addition to allowing traversal, these handles can be used to perform specific transformations which keep the DCEL in a valid state, such as splitting an existing edge in half, creating a new region on the twin of an edge, and removing edges.

Sweep Line Algorithm

Sweep line algorithms are mainly an exercise in bookkeeping; edges must be sorted and traversed in the correct order, and it is necessary to keep a list of which ones are ‘active’ at the current point in time. Removing these details from QuickOpp makes it clearer exactly how the sweep algorithm will function and helps to differentiate between the changes made to sweep over a polygon and the changes made as part of QuickOpp.

For the most part, the polygon sweep line algorithm follows a standard sweep line as you might use for the ‘all segments intersections’ algorithm; but it does have some minor modifications to take into account the ordering of segments on the shape and traverse them in the order they are connected.

QuickOpp

The core algorithm of this library is split into multiple pieces internally, using virtual interfaces to bridge the gaps between them. The individual pieces are the following

- The cost function
- Cost calculation of regions and edges
- Decomposition
- Merging regions

- Forming swaths and boundaries from the merged region groups

The rationale behind splitting the algorithm into these pieces is two-fold:

1. These are distinct steps in processing; creating interfaces between them enforces that they remain separated in their implementation.
2. Defining the algorithm this way allows for experimentation. It is simple to swap out a different decomposition algorithm or merge step by re-implementing the interface. This allows the algorithm to evolve over time for a user's own personal needs.

While the cost function is currently the only piece of the algorithm exposed as an input to the QuickOpp function, it is a simple step to break the algorithm apart and perform the steps individually (in fact, this is how the UI is implemented).

UI Application

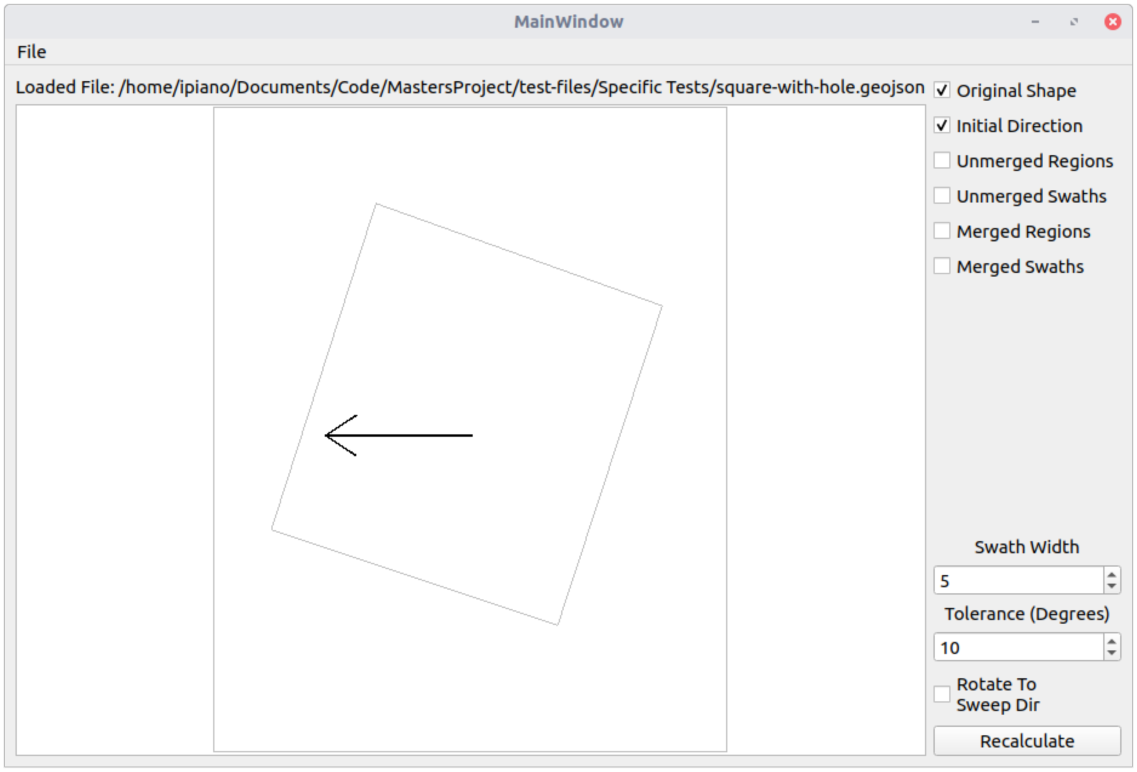
The application developed along with the library allows users to load shapes and run the QuickOpp algorithm. Instead of calling the main QuickOpp function, it utilizes the separate pieces of the algorithm in order to get access to the intermediate steps. This allows the tool to display a number of useful pieces of information:

- The initial direction chosen
- The original shape, rotated with respect to the sweep line
- The regions produced after the initial decomposition step
- Swaths produced to cover the intermediate regions

- The regions after being merged
- Swaths produced to cover the final result

The UI also allows the user to tweak a couple of the input parameters, like the maximum angle between edges in the same region and the width of the machine.

The application is built using the Qt library. This provides a framework which was leveraged to build the UI quickly, without needing to worry about the details of drawing individual items to the screen. Additionally, the Qt Plugins system was utilized to make the application extensible in what input formats it can consume. The initial project includes a plugin which can consume GeoJSON (or geojson) formatted files, but implementing a plugin for other file types would be simple to do with Qt Plugins system. Geojson is a JSON schema specifically for describing geometric figures which is described in RFC 7946. The UI will only accept a subset of the standard; files loaded must contain exactly one Polygon object which may or may not have interior holes. Images of the UI being used to load and process a shape can be seen in figures 5, 6, and 7.

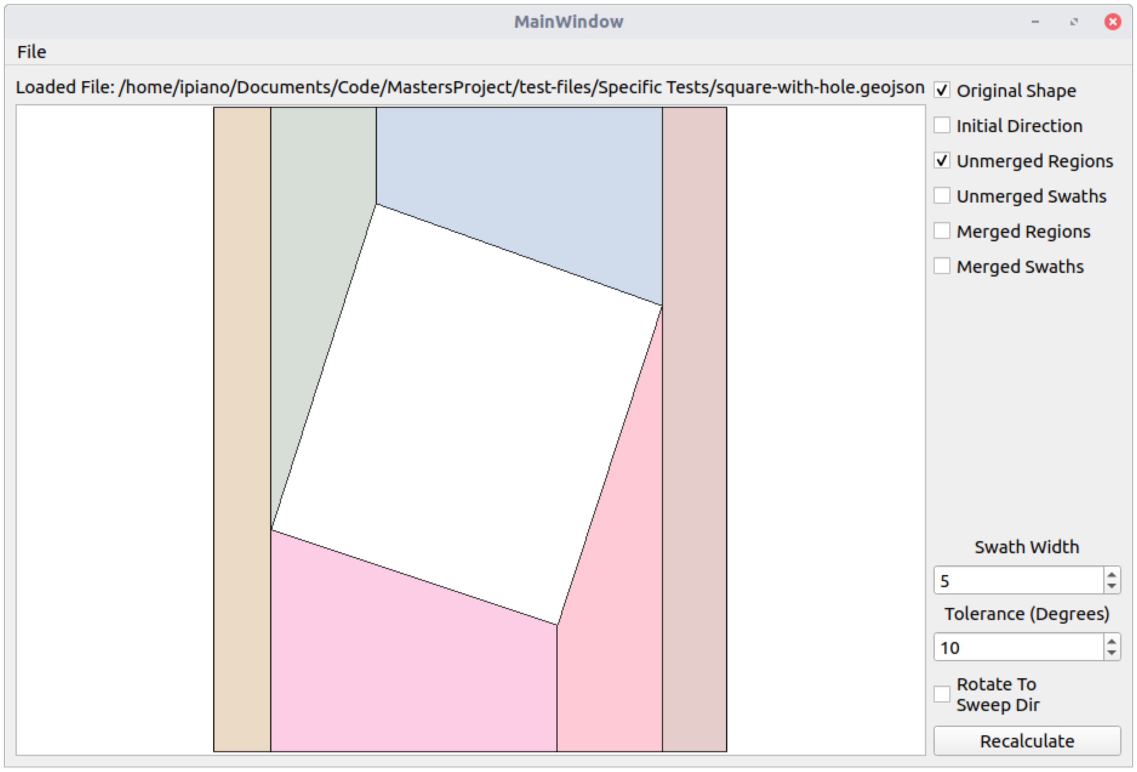


Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Specific Tests/square-with-hole.geojson

[✓] Original Shape
[✓] Initial Direction
[ ] Unmerged Regions
[ ] Unmerged Swaths
[ ] Merged Regions
[ ] Merged Swaths

Swath Width: 5
Tolerance (Degrees): 10
[ ] Rotate To Sweep Dir

Figure 5. Loading a shape and viewing the initial direction


MainWindow

File

Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Specific Tests/square-with-hole.geojson

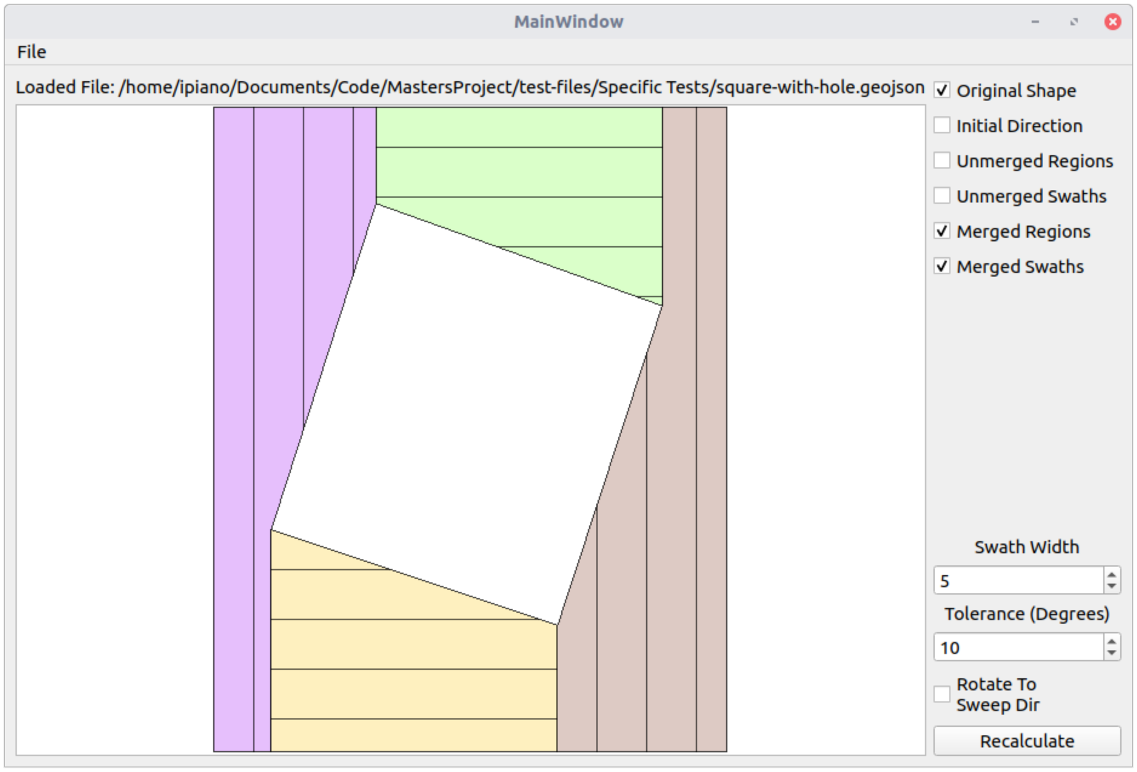
- [✓] Original Shape
- [ ] Initial Direction
- [✓] Unmerged Regions
- [ ] Unmerged Swaths
- [ ] Merged Regions
- [ ] Merged Swaths

Swath Width: 5

Tolerance (Degrees): 10

- [ ] Rotate To Sweep Dir

Figure 6. The final result after merging regions and adding the swaths to travel on.



Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Specific Tests/square-with-hole.geojson

- [✓] Original Shape
- [ ] Initial Direction
- [ ] Unmerged Regions
- [ ] Unmerged Swaths
- [✓] Merged Regions
- [✓] Merged Swaths

Swath Width: 5

Tolerance (Degrees): 10

- [ ] Rotate To Sweep Dir

Figure 7. The regions produced during the decomposition step. You can see that the shape was split where the inner hole was first and last encountered, as well as at the sharp angles it has.

Unit Tests

The googletest unit testing framework was used to develop tests for some of the internal pieces of the algorithm. In particular, there are tests for validating the DCEL structure, the U-Shaped turn cost, and the polygon decomposition algorithm. The tests for the DCEL ensure that a DCEL can be constructed and modified and that the internal representation will remain consistent, the tests for the turning cost verify that it produces a function of the expected shape, and the tests for the polygon decomposition test that a large range of shapes can be decomposed.

The sweep-line algorithm and the QuickOpp algorithm itself are not unit tested. These two pieces of the project have a less well-defined ‘correct’ output, and were constantly being tweaked during development to produce the expected results.

As polygon decomposition is the core of the algorithm, that piece of the algorithm has the most comprehensive set of tests. They validate that it can be used with various combinations of concavities and convexities on both outer and inner shapes. Figure 8 gives an example of some of the inputs used to test it.

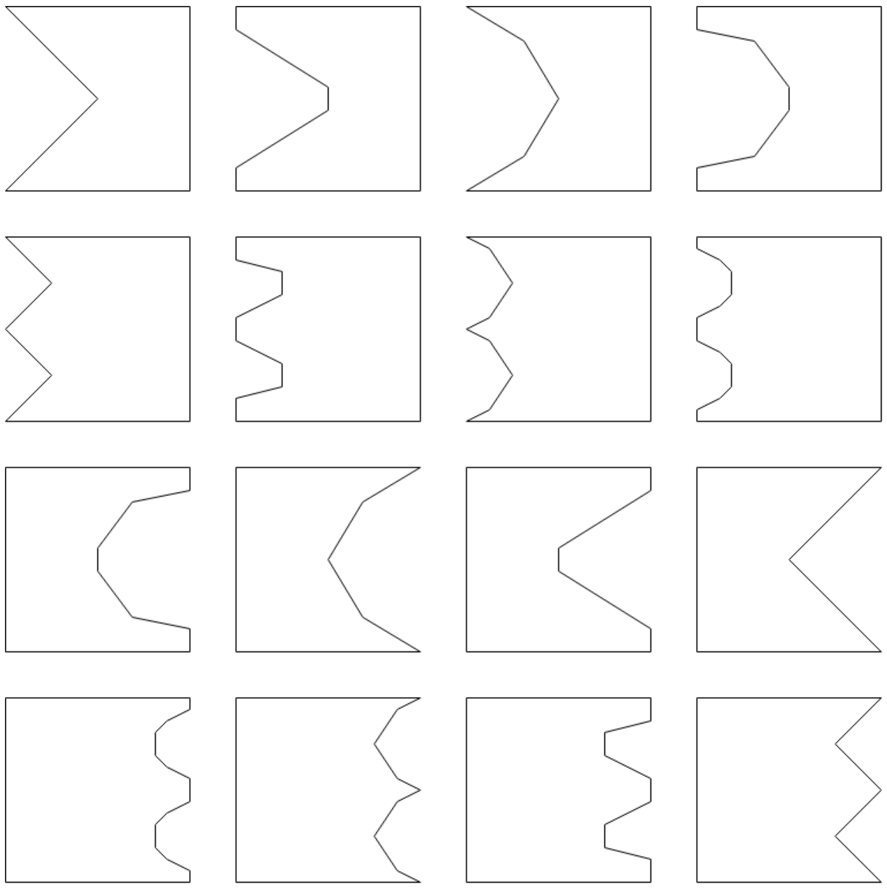


Figure 8. Examples of the test cases for decomposition of an outer loop

Modifications to the Algorithm

Without any reference code from the original project, it is difficult to say how closely this implementation matches the original algorithm. While Driscoll's work does include a full description of the algorithm, there are pieces in the description that are missing or which do not make sense without some context that is not provided; however, it was clear enough that this implementation could be created, and there have certainly been some intentional modifications made.

The most notable modification is in the implementation of the decomposition step. Driscoll began with a classic trapezoidal decomposition, as described by Oksanen (2007) and allowed the decomposition to avoid splitting the shape in a situation where the adjacent edges did not have a sharp enough angle between them. While this may be more efficient, as there is only a single decomposition step, it is more difficult to test the decomposition step, as it takes into account an extra parameter for the angle of subdivision. This implementation uses a two-step decomposition, where the first step is an exact cellular decomposition as researched by Choset and Pignon (1997), Choset (2000), and Atkar et al. (2001). This decomposition, sometimes called a boustrophedon decomposition, only splits the shape at the points where an inner hole is encountered for the first or last time. This produces much larger regions which may not be convex on the top or bottom. After this initial decomposition is performed, a second pass is made to further divide these regions based on the relative angle of adjacent edges, producing a final decomposition similar to that of Driscoll's algorithm.

Another significant modification is in the DCEL produced by the decomposition step. Driscoll makes a point of showing that any given region can only have at most four adjacent neighbors (two on each side). It is unclear exactly how this is achieved with QuickOpp, but it appears to be the result of how QuickOpp constructs the DCEL during the decomposition step. Logically, regions may have more than two adjacent regions on either side, as figure 9 demonstrates. Since this implementation manages the DCEL construction wholly separately from the decomposition step, the end result of decomposition no longer guarantees that a region will have a maximum of four neighbors. However, the recursive merge step used to re-combine them was not difficult to modify to handle this situation.

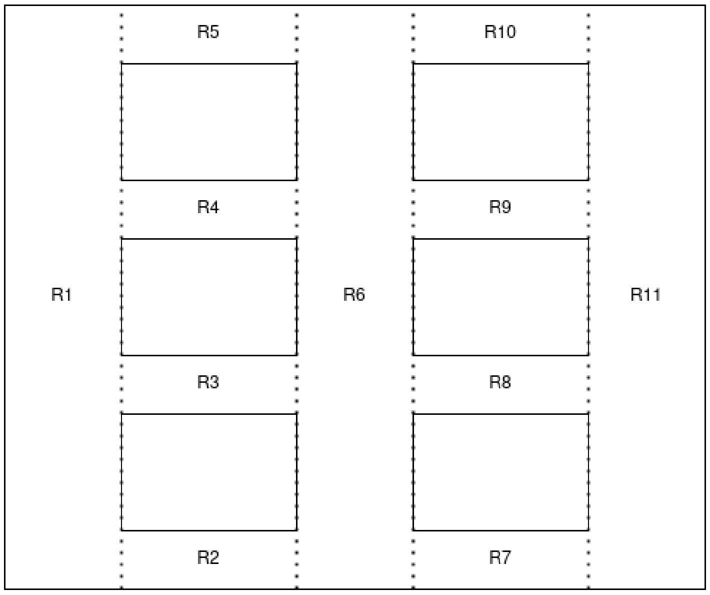
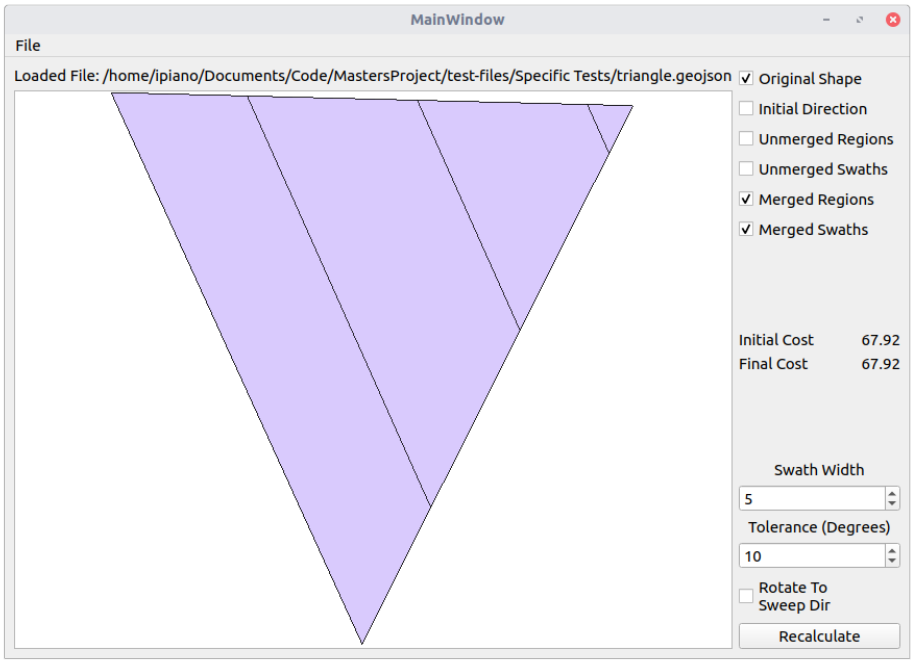


Figure 9. Example of how regions could have more than 4 adjacent regions. It is unclear how QuickOpp should produce Region 6 with at most 4 adjacent regions

Outcomes

In a perfect world, this section would contain side-by-side comparisons of the results given by this project and the original QuickOpp algorithm as well as examples of using the algorithm to plan against some real-world recorded data. An attempt was made with no success to contact Theresa Driscoll when the project was begun, but no response was received, so there is no way to acquire her test data for comparison. There were also plans initially that some work on this project would be sponsored by Raven Industries Inc. but those plans were never realized, so acquiring data recorded by real equipment has been difficult.

However, it is still possible to show that this implementation is at least producing solutions which have a lower cost than the initial input. Looking at some of the trivial cases that were used for initial development, it can be seen that, when there is no way to partition the field to produce a better result, this project does not produce a result that is worse than the initial solution. This can be seen in figures 10-12. (Take note of the 'inital cost' and 'final cost' values on the right side of the images).



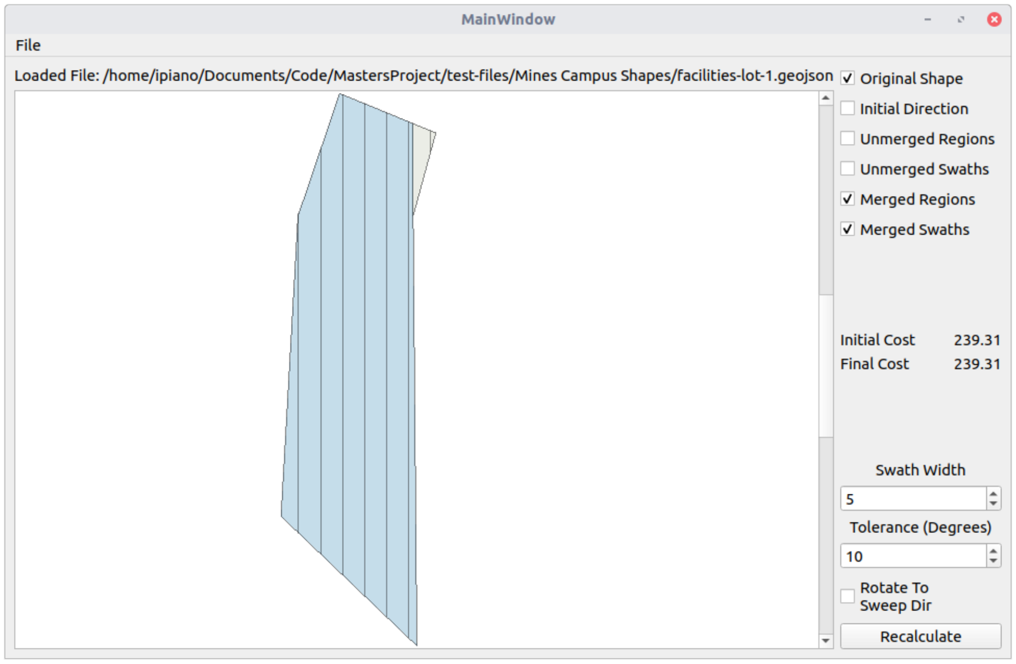
Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Specific Tests/triangle.geojson

- [✓] Original Shape
- [ ] Initial Direction
- [ ] Unmerged Regions
- [ ] Unmerged Swaths
- [✓] Merged Regions
- [✓] Merged Swaths

Initial Cost	67.92
Final Cost	67.92

Swath Width: 5
Tolerance (Degrees): 10
[ ] Rotate To Sweep Dir

Figure 10. Simple case: A convex shape with no sub-regions



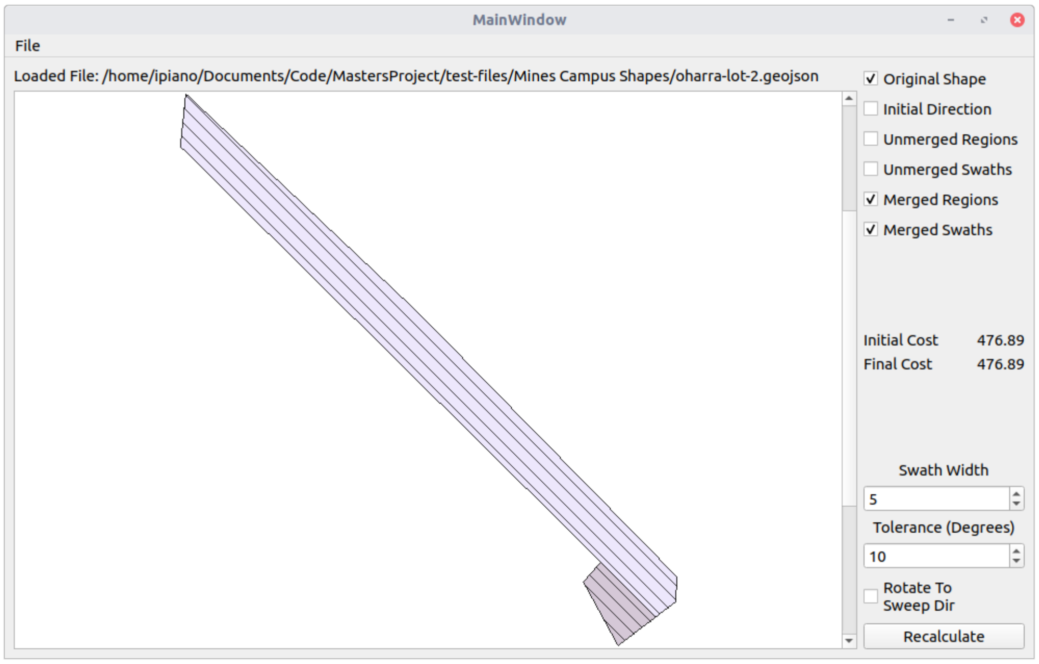
Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Mines Campus Shapes/facilities-lot-1.geojson

- [✓] Original Shape
- [ ] Initial Direction
- [ ] Unmerged Regions
- [ ] Unmerged Swaths
- [✓] Merged Regions
- [✓] Merged Swaths

Initial Cost	239.31
Final Cost	239.31

Swath Width: 5
Tolerance (Degrees): 10
[ ] Rotate To Sweep Dir

Figure 11. Simple case: A shape with a single sub-region



Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Mines Campus Shapes/oharra-lot-2.geojson

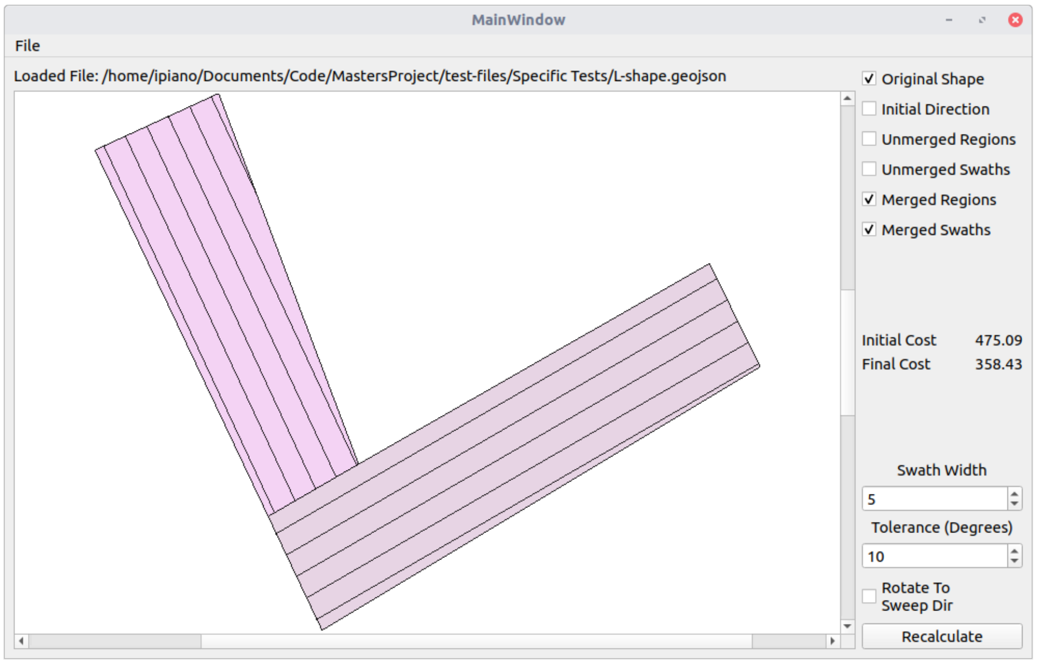
[✓] Original Shape
[ ] Initial Direction
[ ] Unmerged Regions
[ ] Unmerged Swaths
[✓] Merged Regions
[✓] Merged Swaths

Initial Cost	476.89
Final Cost	476.89

Swath Width: 5
Tolerance (Degrees): 10
[ ] Rotate To Sweep Dir

Figure 12. Simple case: A shape with a single sub-region

Similarly, in cases where there is an obviously better way to cover the field, a solution is found which lowers the overall cost. As can be seen in figures 13-15 this does not always find the *best* solution; but this is to be expected, as the algorithm is designed with heuristics that work best with common farming practices and these examples are not shapes that would be found in real fields.



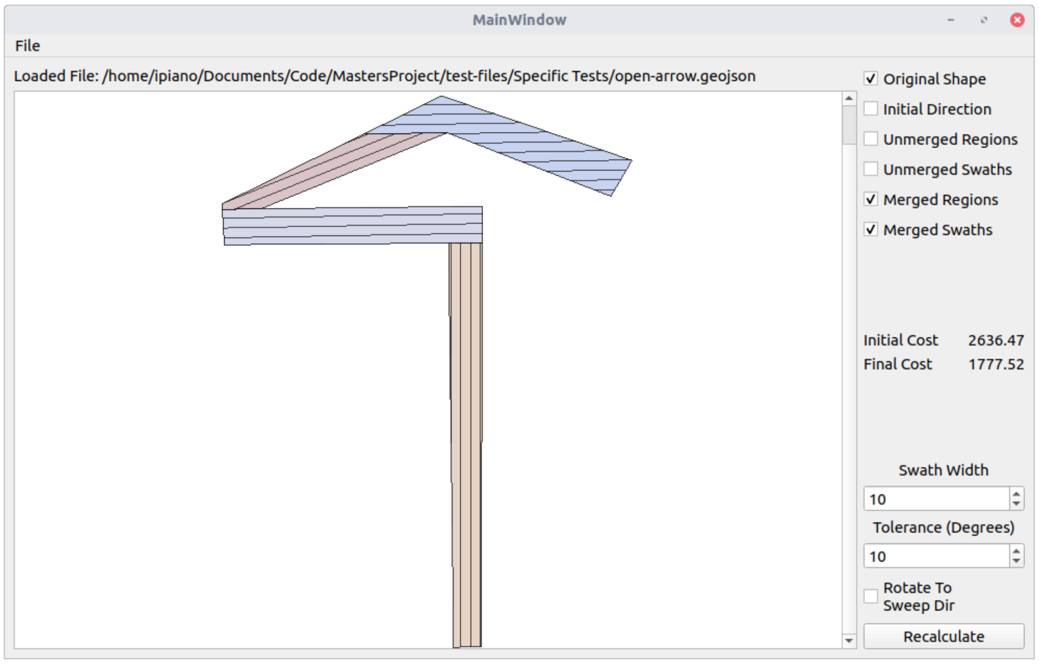
Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Specific Tests/L-shape.geojson

☑ Original Shape
☐ Initial Direction
☐ Unmerged Regions
☐ Unmerged Swaths
☑ Merged Regions
☑ Merged Swaths

Initial Cost	475.09
Final Cost	358.43

Swath Width: 5
Tolerance (Degrees): 10
☐ Rotate To Sweep Dir

Figure 13. Nontrivial case: An 'L' shape, where one arm of the field should be covered perpendicular to the other. Cost reduction: ~24%



Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Specific Tests/open-arrow.geojson

- [✓] Original Shape
- [ ] Initial Direction
- [ ] Unmerged Regions
- [ ] Unmerged Swaths
- [✓] Merged Regions
- [✓] Merged Swaths

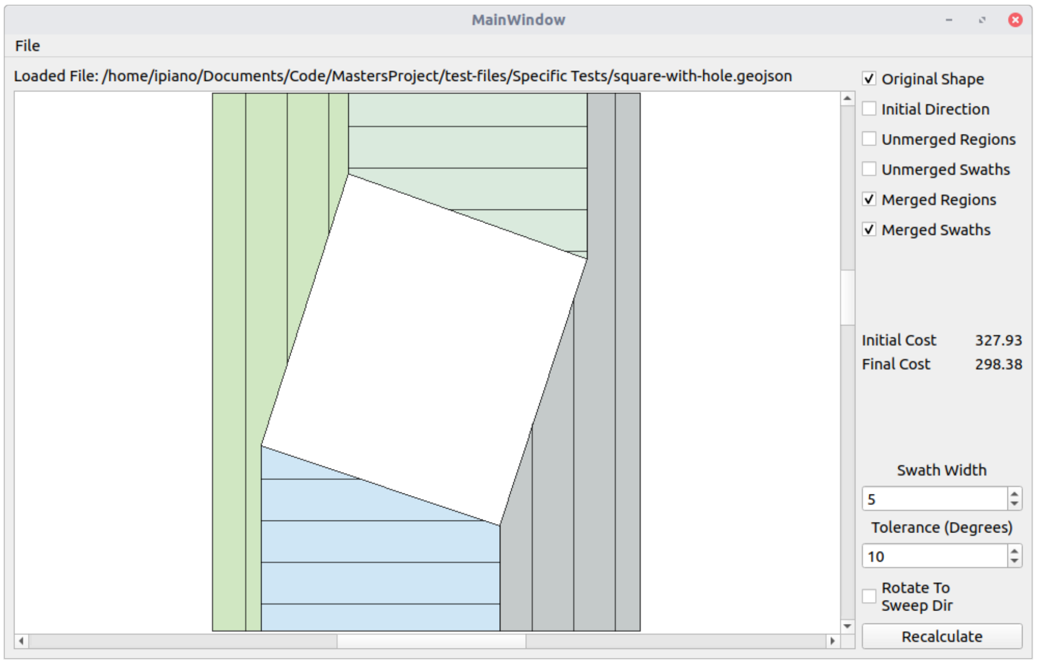
Initial Cost	2636.47
Final Cost	1777.52

Swath Width: 10

Tolerance (Degrees): 10

- [ ] Rotate To Sweep Dir

Figure 14. Nontrivial case: An open arrow shape. Notice that the top-most region could potentially be better covered at an angle, but this was not detected by the algorithm. Cost reduction: ~32%



Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Specific Tests/square-with-hole.geojson

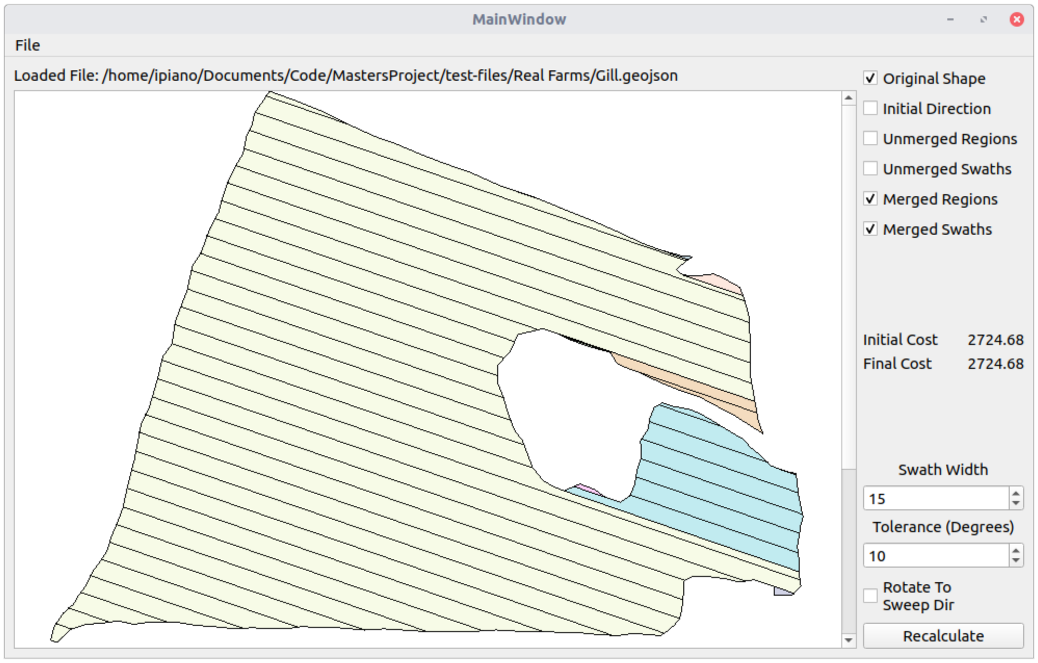
✓ Original Shape
Initial Direction
Unmerged Regions
Unmerged Swaths
✓ Merged Regions
✓ Merged Swaths

Initial Cost	327.93
Final Cost	298.38

Swath Width: 5
Tolerance (Degrees): 10
Rotate To Sweep Dir

Figure 15. Nontrivial case: A square with a rotated square hole in the middle. Cost reduction: ~9%

Furthermore, a number of test cases were created by drawing field boundaries manually using the online tool 'geojson.io'. These cases were created by finding farms on a satellite map and tracing around the outer and inner shapes with the editor. While the shapes do not accurately represent what might be recorded by a vehicle with high-precision GPS, they do demonstrate that the algorithm can be used successfully for more realistic shapes. As figures 16-18 show, most of these fields result in a single direction for the majority of the field with a few small pieces that should be planted in the direction normal to that.



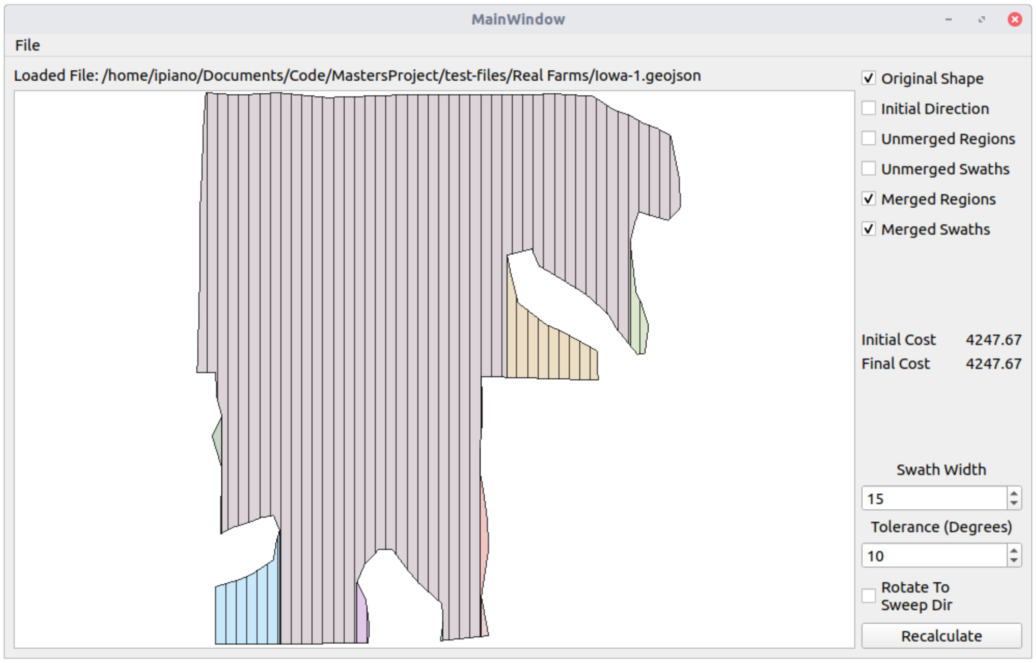
Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Real Farms/Gill.geojson

- ✓ Original Shape
- Initial Direction
- Unmerged Regions
- Unmerged Swaths
- ✓ Merged Regions
- ✓ Merged Swaths

Initial Cost	2724.68
Final Cost	2724.68

Swath Width: 15
Tolerance (Degrees): 10
Rotate To Sweep Dir

Figure 16. Real world example. No cost reduction



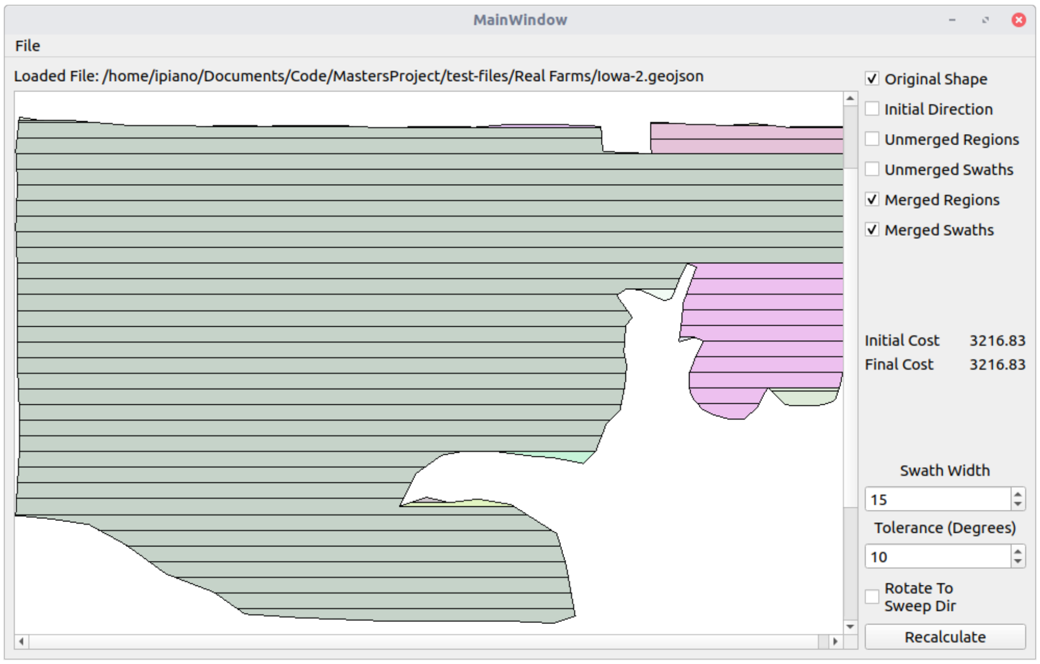
Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Real Farms/Iowa-1.geojson

- ✓ Original Shape
- Initial Direction
- Unmerged Regions
- Unmerged Swaths
- ✓ Merged Regions
- ✓ Merged Swaths

Initial Cost	4247.67
Final Cost	4247.67

Swath Width: 15
Tolerance (Degrees): 10
Rotate To Sweep Dir

Figure 17. Real world example. No cost reduction



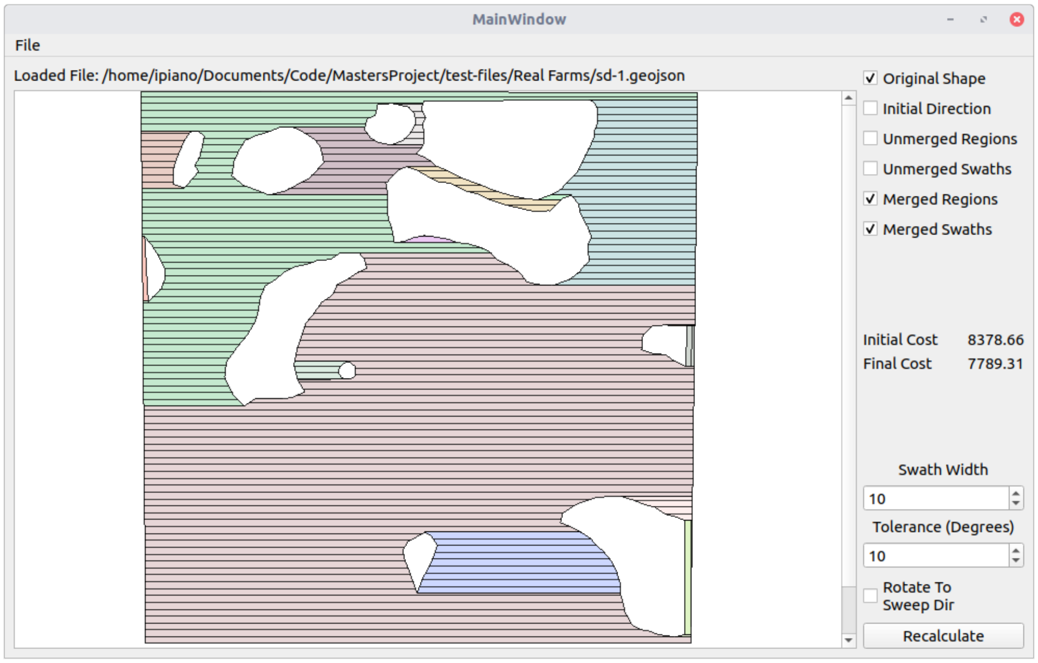
Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Real Farms/Iowa-2.geojson

✓ Original Shape
Initial Direction
Unmerged Regions
Unmerged Swaths
✓ Merged Regions
✓ Merged Swaths

Initial Cost 3216.83
Final Cost 3216.83

Swath Width 15
Tolerance (Degrees) 10
Rotate To Sweep Dir

Figure 18. Real world example. No cost reduction



Loaded File: /home/ipiano/Documents/Code/MastersProject/test-files/Real Farms/sd-1.geojson

✓ Original Shape
Initial Direction
Unmerged Regions
Unmerged Swaths
✓ Merged Regions
✓ Merged Swaths

Initial Cost 8378.66
Final Cost 7789.31

Swath Width 10
Tolerance (Degrees) 10
Rotate To Sweep Dir

Figure 19. Real world example. Cost reduction: ~7%

It is apparent from these examples that the algorithm is best suited to data produced from real fields, and it is regrettable that none could be obtained for testing. However, it is evident that the implementation of QuickOpp is at the very least functional, and it could be easily modified to suit the needs of someone who wanted to use it in a more specific context.

Further work on this project should begin by testing against some real-world data in order to verify correctness. Following that, a number of improvements could potentially be made by adding a post-processing step. While it may not be particularly relevant in real scenarios, situations like that shown in figure 14 could be improved by testing each of the final regions for a best overall direction, to see if a better one could be found than the one produced by the merge step. This has the potential to provide better results because the merge algorithm defined for QuickOpp does not take into account every possible direction when considering the merge of two regions; it only considers the best-case directions for each region individually and the directions perpendicular.

Citations

Atkar, Prasad N., et al. "Exact cellular decomposition of closed orientable surfaces embedded in R^3." *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164).* Vol. 1. IEEE, 2001.

Choset, Howie. "Coverage of known spaces: The boustrophedon cellular decomposition." *Autonomous Robots* 9.3 (2000): 247-253.

Choset, Howie, and Philippe Pignon. "Coverage path planning: The boustrophedon cellular decomposition." *Field and service robotics.* Springer, London, 1998.

Driscoll, Theresa Marie. "Complete coverage path planning in an agricultural environment." (2011).

Jin, Jian. "Optimal field coverage path planning on 2D and 3D surfaces." (2009).

Khan, Amna, Iram Noreen, and Zulfiqar Habib. "On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges." *J. Inf. Sci. Eng.* 33.1 (2017): 101-121.

Meuth, Ryan J., and Donald C. Wunsch. "Divide and conquer evolutionary TSP solution for vehicle path planning." *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008.

Oksanen, Timo, and A. Visala. "Path planning algorithms for agricultural machines." *Agricultural Engineering International: CIGR Journal* (2007).

Zelinsky, Alexander, et al. "Planning paths of complete coverage of an unstructured environment by a mobile robot." *Proceedings of international conference on advanced robotics*. Vol. 13. 1993.