University of Athens
Dept. of Informatics & Telecommunications

K17c: Software Development for Algorithmic Problems

Kyriakopoulos Dimitrios* – `sdi1900093`
Poulis Angelos* – `sdi1900230`
Rontogiannis Dimitrios* – `sdi1900165`

Fall 2022 - Project 1 (Point-set Polygonization)

Contents

1 Introduction

We implemented three algorithms (Incremental/Based on the convex hull/Onion) for point set polygonization and Pick's theorem for calculating the area of the polygon. We provide the aforementioned implementations along with bash scripts for benchmarking the algorithms and a Python script to visualize the results. Our code was written in C++, using the CGAL library. At the end of this report, we present an experimental study of the algorithms on a sampled set of cases.

*All authors have contributed equally.

2 What is included

```
Point-Set-Polygonization
├── build - files created by CMake
├── CMakeLists.txt - CMake for build
├── include - Header files
│   ├── ch2polyline_algo.hpp - Definition of based on CH algorithm
│   ├── common.hpp - Definition of some common used functions
│   ├── graham_scan.hpp - Definition of Graham Scan
│   ├── incremental_algo.hpp - Definition of Incremental algorithm
│   ├── onion_algo.hpp - Definition of Onion algorithm
│   ├── pick_algo.hpp - Definition of Pick's theorem
│   └── utils.hpp.hpp - Definition of util functions
├── src - Source files
│   ├── ch2polyline_algo.cpp - Implementation of based on CH algorithm
│   ├── graham_scan.cpp - Custom Graham Scan algorithm
│   ├── incremental_algo.cpp - Implementation of Incremental algorithm
│   ├── main.cpp - Main function implmentation
│   ├── onion_algo.cpp - Implementation of Onion algorithm
│   ├── pick_algo.cpp - Implementation of Pick's theorem
│   └── utils.cpp - Implementation of util functions
└── scripts - Helper scripts
    ├── plot_graphs.py - Python script for plotting benchmarks
    ├── run_test_cases.sh - Bash script for running test-cases
    └── run_benchmarks.sh - Bash script for benchmarking the algorithms
```

3 Usage

3.1 Build and run

CMake file it is provided for compilation of the project. Build and run the project as follows. Go to the /build directory and run:

```
$ cmake .. && make
$ ./bin/to_polygon -i <input_file> -o <output_file>
-algorithm <incremental/convex_hull/onion>
-edge_selection <1 to 3 | except onion>
-initialization <1a/1b/2a/2b | only on incremental>
-onion_initialization <1 to 5 | only on onion>
```

Parameters description:

- <input_file> : File with the point-set to polygonize.
- <output_file> : File to print the results.
- <algorithm> : Polygonization algorithm to run:
  - "incremental" for Incremental
  - "convex_hull" for Based on CH
  - "onion" for Onion
- <edge_selection> : Visible edge selection policy (not in onion algorithm):
  1. Random
  2. Adds minimal area
  3. Adds maximal area
- <initialization> : Initialization policy (only for incremental algorithm):
  1. Sort by x coord.
     (a) descending
     (b) ascending
  2. Sort by y coord.
     (a) descending
     (b) ascending
- <onion_initialization> : Onion initialization policy (only for onion algorithm):
  1. Random selection
  2. Ascending x-coord.

3. Descending x-coord.
4. Ascending y-coord.
5. Descending y-coord.

3.2 Testing scripts

We also provide some scripts for testing. Run them as follows from the scripts/ directory:

```
$ ./test_them_all.sh -i <test cases folder>
                     [-o <output_folder> | optional]
$ ./run_test_cases -i <test cases folder>
    -algorithm <incremental/convex_hull/onion>
    -initialization <1a/1b/2a/2b | only in incremental>
    -edge_selection <1 to 3 | except onion>
    -onion_initialization <1 to 5 | only in onion>
```

- test_them_all.sh - *Runs every algorithm with every option for all test cases (using the run_test_cases.sh). Optionally, prints the number of points and the elapsed time for each test case, in the output folder provided (using -o parameter).*
- run_test_cases.sh - *Runs a given algorithm with its options for all test cases in the specified folder.*

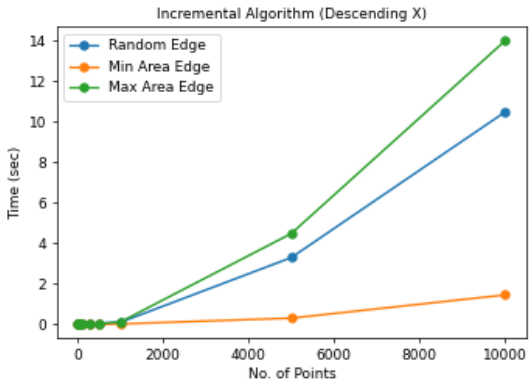
In addition, we provide a python script that plots the time to points for each possible combination of options of each algorithm, for the sampled test cases. The points are on the x-axis and the time on the y-axis. The folder containing the benchmark times for the graph must be given as an argument. Go to the scripts/ directory and run:

```
$ python3 plot_graphs.py <input_folder>
```

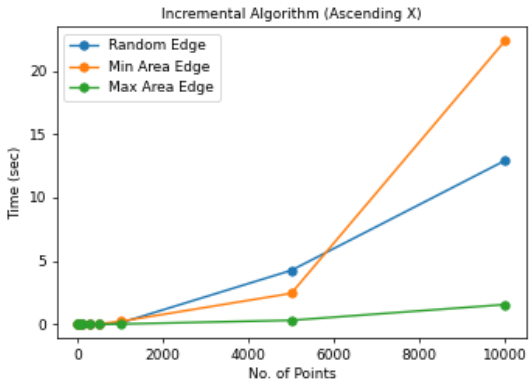
4 Experiments

We conducted a set of experiments on ten *sampled* instances, five from each of the *images* and *uniform* folders. Specifically, we tested the following instances:

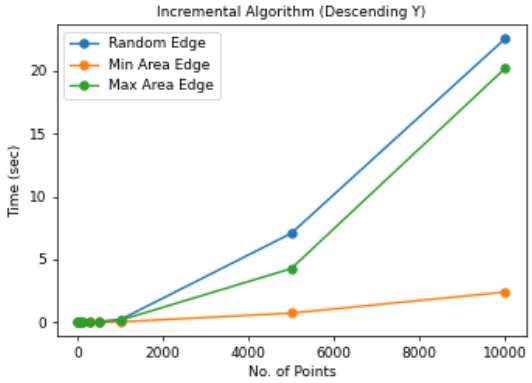
```
instances/data/images/uniform-0000010-1.instance
instances/data/images/uniform-0000020-1.instance
instances/data/images/uniform-0000030-2.instance
instances/data/images/uniform-0000050-1.instance
instances/data/images/uniform-0000100-2.instance
instances/data/images/euro-night-0000300.instance
instances/data/images/euro-night-0000500.instance
instances/data/images/euro-night-0001000.instance
instances/data/images/euro-night-0005000.instance
instances/data/images/euro-night-0010000.instance
```



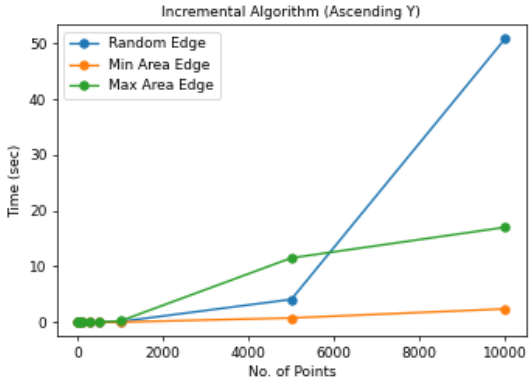
(a) Descending X



(b) Ascending X



(c) Descending Y



(d) Ascending Y

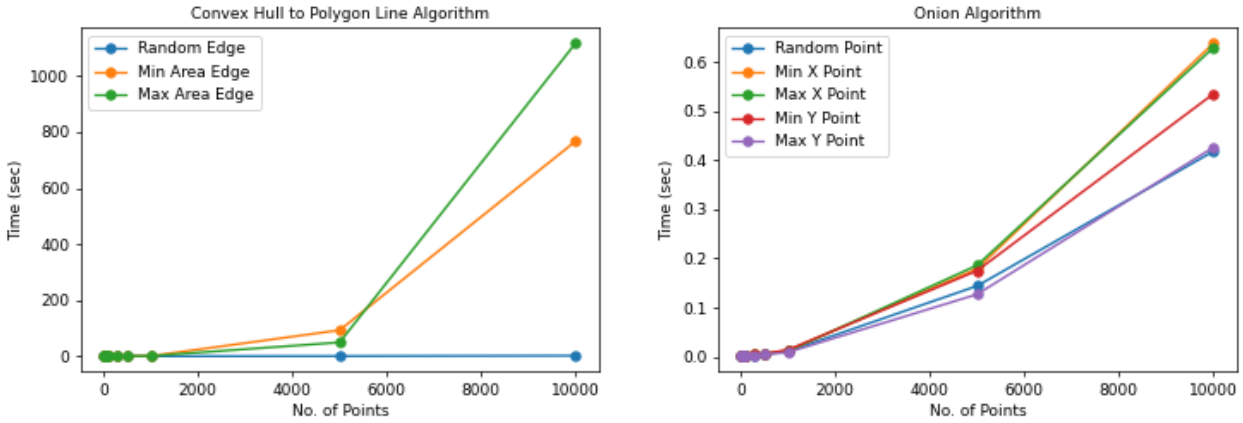
Figure 1: Incremental algorithm with X/Y ascending/descending options

Figures 1(a) & 1(c) shows that the incremental algorithm with points sorted in descending order with respect to X/Y, performs better with the *min area edge* option. In contrast, the *random edge* and *max area edge* selection options show an exponential rise in time. Although, this test was conducted in a limited setting and hence we cannot conclude on the trend.

Figure 1(b) shows that the incremental algorithm with points sorted in ascending order with respect to X has the best performance with the *max area edge* selection option. The *min area edge* option, on the other hand, shows a rapid increase in time, while the *random edge* option lies between the other two.

Figures 1(d) shows that the incremental algorithm with points sorted in ascending order with respect to Y performs better with the *min area edge* selection option. The *random edge* option, on the other hand, shows a rapid increase in time, while the *max area edge* option is in the interim. In conclusion, we see that the incremental algorithm is sensitive to the edge selection policy and the parameters of the initialization options.

We ran all three algorithms for each possible combination of options on all instances.



(a) Based on CH algorithm

(b) Onion algorithm

Figure 2: Based on CH and Onion algorithms

Figure 2(a) shows that algorithm Based on Convex Hull, performs best with the *random edge* selection option. In contrast, the *min/max area edge* selection options show an exponential trend in time.

Figure 2(b) shows that the Onion algorithm has almost the same performance with any initialization option, which makes it the most robust to initialization parameters of the three algorithms.

5 Conclusion

Based on our experiments, we can say that in general, the onion algorithm was the most efficient among the three, while the other two (incremental & based on convex hull) have similar performance.