# Implementation of a fast and efficient hull algorithm.


UPPSALA UNIVERSITET

Project form:
Schlumberger
Information Solutions,
Oslo, Norway

Michelle Brundin
mibr8123@student.uu.se

Emil Jansson
emja7157@student.uu.se

Emil Rosén
emro6415@student.uu.se

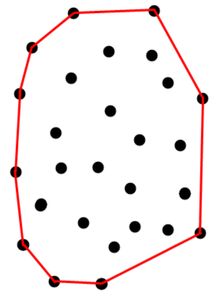
Supervisors:
Tanya Moguchaya
Schlumberger
Victor Aarre
Schlumberger

Department of
Information Technology
Box 337
SE-751 05 Uppsala
Sweden

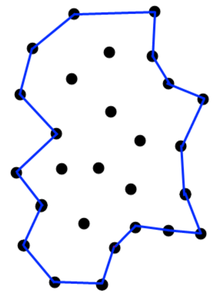


## Summary

We have developed and implemented a fast algorithm for calculating the concave hull of a set of points in 2D. The algorithm is implemented in c++ and the performance of the algorithm satisfies the requirements. It can calculate the concave hull for a set of 10 000 000 points in under 10 seconds on a Intel Core 2 duo PC.

Convex Hull



Concave hull

## Aims

The goal was to implement a concave hull algorithm that is:

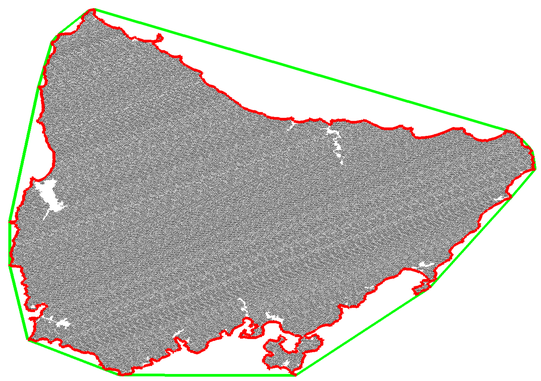
- Fast: 10 000 000 in under 10 seconds
- Reliable: Algorithm should never fail.
- Easy to use: No input parameters required.

## The Algorithm

- Calculate the convex hull
  - Many known algorithms
- Convert the convex hull to a concave hull
  - We developed the Gift Opening algorithm

### Gift Opening

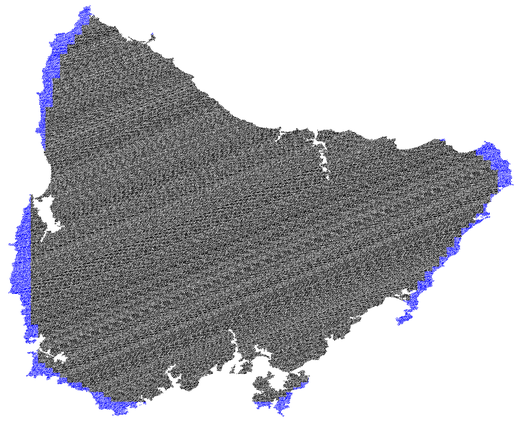
The Gift Opening algorithm converts a convex hull to a concave hull. It splits the edges in the convex hull and finds new points that can be part of the concave hull.



The convex (–) and concave (–) hull of a set of 300 000 points

## Optimizations

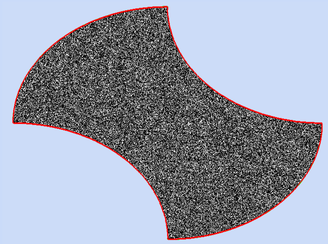
Both the convex and the concave algorithms have been very optimized. The biggest optimization is that the points are divided into boxes and only the relevant boxes are used in the calculations. Many parts of the algorithm has also been parallelized to run on more than one CPU.



Optimization: Only Blue points are used when calculating convex hull.

## Results

We measured the time for the algorithm to complete on a test case meant to test both the convex and concave part of the algorithm. The algorithm was tested between 1 000 000 and 16 000 000 points to test speed and scaling.



*The test case*