# TESTPROTOCOL

Program: Master AI-Engineering
Lecture: Advanced Programming

# Assignment: Convex Hull
# Visualization & Performance Evaluation

Students: Ivan Birkmaier, Herbert Grünsteidl, Leonard Hübner
ID: 2310585049, 2310585012, 2310585040

Lecturer: Alexander Nimmervoll

Wien, 04.10.2023

FACHHOCHSCHULE
TECHNIKUM WIEN

# Table of Contents

# 1 Overview

In the following test protocol, we will discuss the performance of the Quick Hull and Giftwrapping algorithm, when computing the convex hull of a point cloud.

# 2 Methodology

We implemented the algorithms using python. For the computational part we used python native libraries as well as the numpy library. For visualization we used the pygame library.

## 2.1 Time Measurement

We implemented two methods of time measurement:

- Python native Module time, measuring the time of execution in seconds.
- Python native Module cProfile, counting the number of function calls during execution.

## 2.2 Test Cases

To evaluate the performance, we generated 4 different types of point clouds:

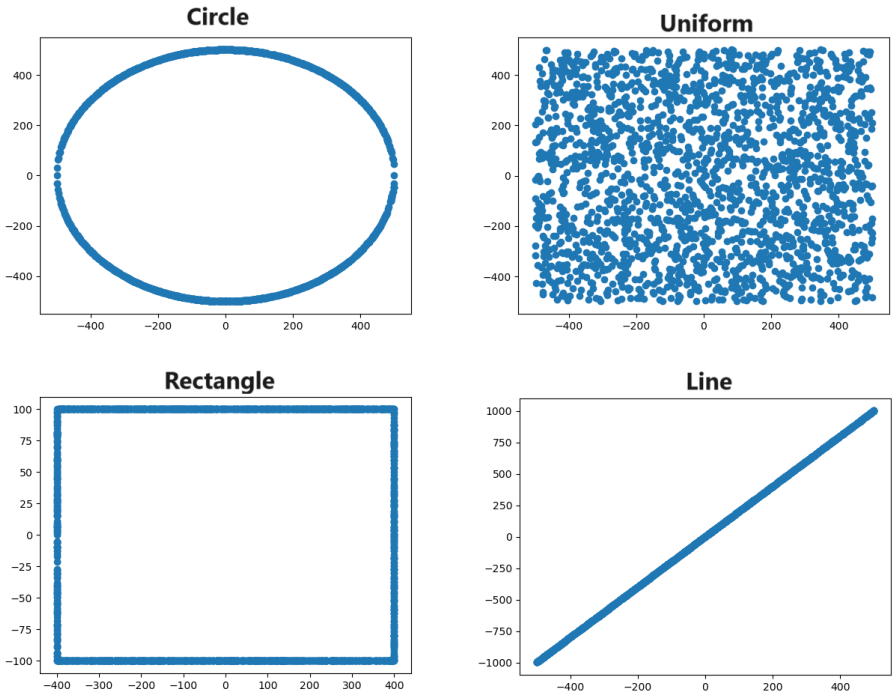


Figure 1: the 4 types of sample point clouds used

## 2.3 Test Runs Special Cases (Circle, Rectangle, Line)

For the special cases we measured performance for N=2^i number of points, for i ranging from 1 to 20, spanning the range from 2 to 1.048576*10^6 points.

## 2.4 Test Runs Uniform

For the uniformly distributed data points we measured performance for N=2^i number of points, for i ranging from 1 to 25, spanning the range from 2 to 3.355*10^7 points.

# 3 Measurements

## 3.1 Test Runs Special Cases

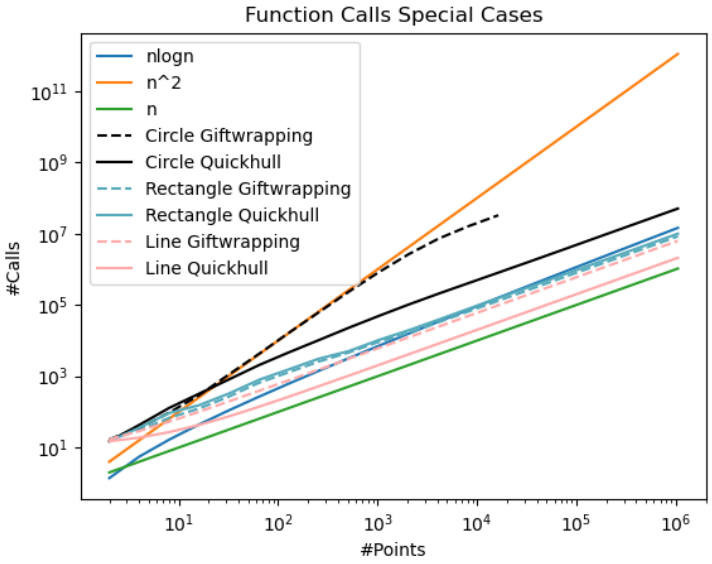


Figure 2: performance measure special cases function calls

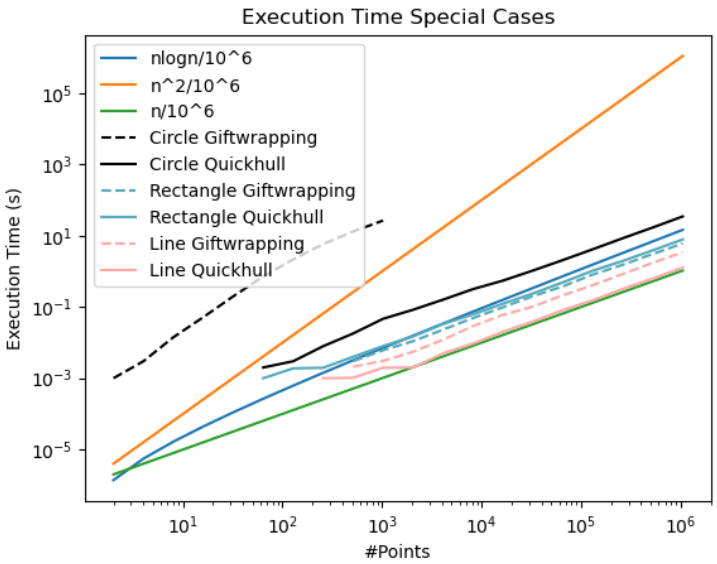


Figure 3: performance measure special cases execution time

In Figure 2 the number of points is plotted against the # function calls the algorithms took to complete the calculation. Furthermore y=nlogn, y=n^2 and y=n are plotted for reference.
Figure 3 show the same runs but the measured runtime instead of # the function calls in seconds. The reference functions are scaled down by 10^-6 to provide useful insights.
For the giftwrapping algorithm we were not able measure beyond 2^15 points because of the square scaling behavior and our limited runtime. In addition, the smallest increment of the runtime is ~1 µs, so we could not plot all series fully.
From Figure 2 and 3 we can conclude that for both algorithms circular distributed data is a worst case, the bigger the data set the more it trends towards nlogn.
Furthermore, we can see both rectangular shaped data as well as line shaped data performs slightly better than nlogn.

## 3.2 Test Runs Uniform Case

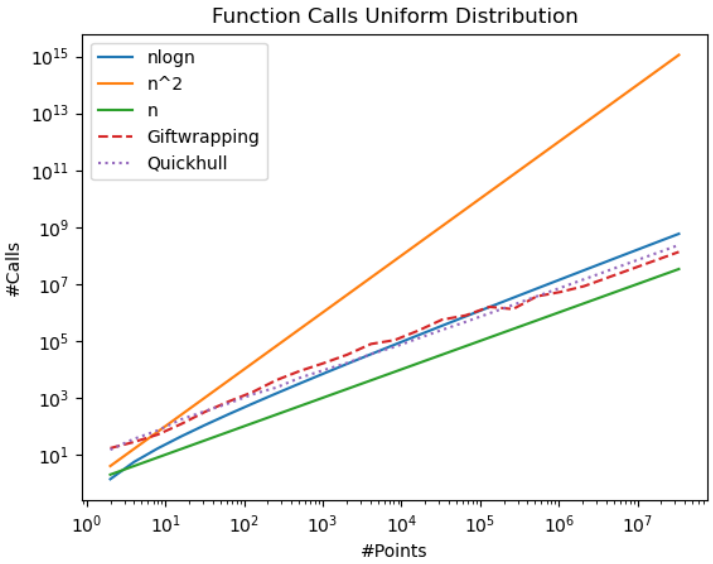


Figure 4: performance measure uniform case function calls

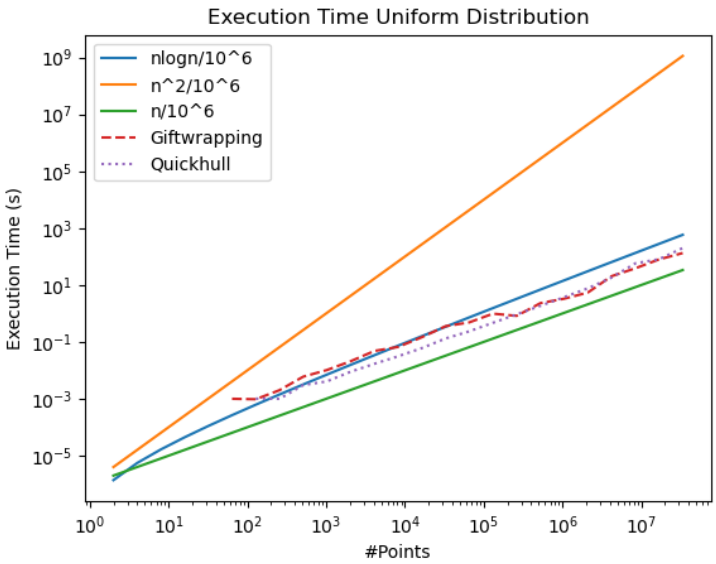


Figure 5: performance measure uniform case execution time

In Figure 4 and 5 we can see the same performance statistics as in Figure 2 and 3.
The performance statistics show us that both algorithms are bound by nlogn for large n.
In our case Quickhull performs better than nlogn at N~5*10^3, Giftwrapping only at N~2*10^5.
When reaching N~5*10^5 Giftwrapping starts outperforming Quickhull regarding function calls.
Looking at the execution time we can see Giftwrapping catching up but not outperforming.
Therefore, our results tell us for an average case and large:

$$Runtime\big(Quickhull(n)\big) = O(nlogn)$$

$$Runtime\big(Giftwrapping(n)\big) = O(nlogn)$$

This corresponds to the literature, Quickhull is bound on average by nlogn because of its nature in dividing up the points recursively, like the divide and conquer algorithm.
For the Giftwrapping Algorithm in theory has a performance of O(nh) where h is the number of points on the hull. This is why the performance varies a lot more for the Giftwrapping algorithm