GPU-accelerated 2D Convex Hull Computation

Nhóm 6

Dương Thanh Hải,
Mai Thanh Hảo,
Tôn Nữ Mai Khanh,
Hoàng Thảo Nguyên

Convex Hull

- Given a set of points P. A convex hull of P is the smallest convex polygon which encloses all points in P.
  - Convex: For any two points p, q inside a convex polygon, the line segment $\overline{pq}$ is completely inside it.
- The problem:
  - Input: Set $P = \{p_1, p_2, ..., p_n\}$.
  - Output: A subset $S \subset P$ consists of points that are vertices of the convex hull in clockwise order.

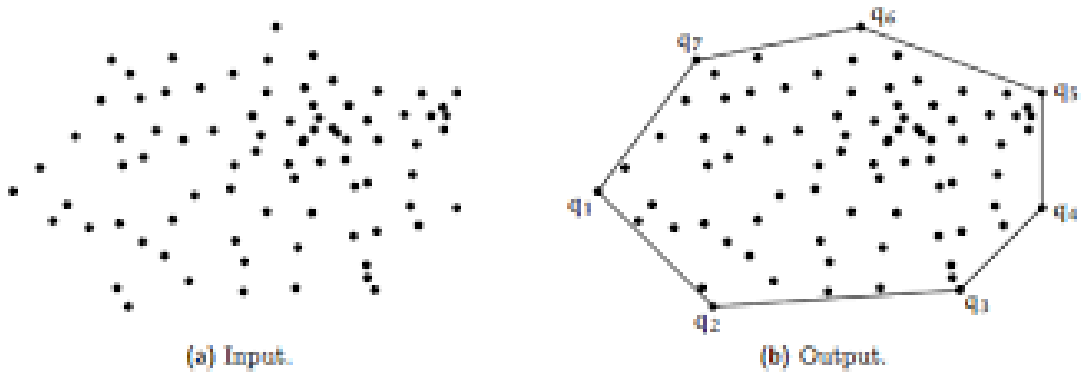


Figure 1: Convex hull

QuickHull Algorithm

- The QuickHull algorithm is a Divide and Conquer algorithm similar to QuickSort.
- Time Complexity: The analysis is similar to Quick Sort. On average, we get time complexity as O($nlogn$), but in worst case, it can become O(n^2)

QuickHull Algorithm

1: {**First Split**}
2: Find an d dimensional face with extrema points. Add these points to the convex hull.
3: Divide the remaining points into two sets: above and below the face.
4: For each point, find distance from point to face.
5: {**Recursive Step**}
6: **for** each set **do**
7: Pick the point with max distance. Add these points to the convex hull.
8: Form a simplex with the points and the face.
9: Test each point for inclusion within the simplex. If it is inside, then it cannot be part of the convex hull. Throw the point out.
10: For each point, find the face of the simplex closest to it. Split into n sets.
11: For each point, compute the distance from the point to the face.
12: Repeat until all points have been processed.

Figure 2: Pseudocode of QuickHull Algorithm

QuickHull Algorithm using MultiProcessing [1]

- We will create two processes and each process respons for computing a portion of the convex hull.
- After that, we will combines the results for output.

QuickHull Algorithm using MultiProcessing

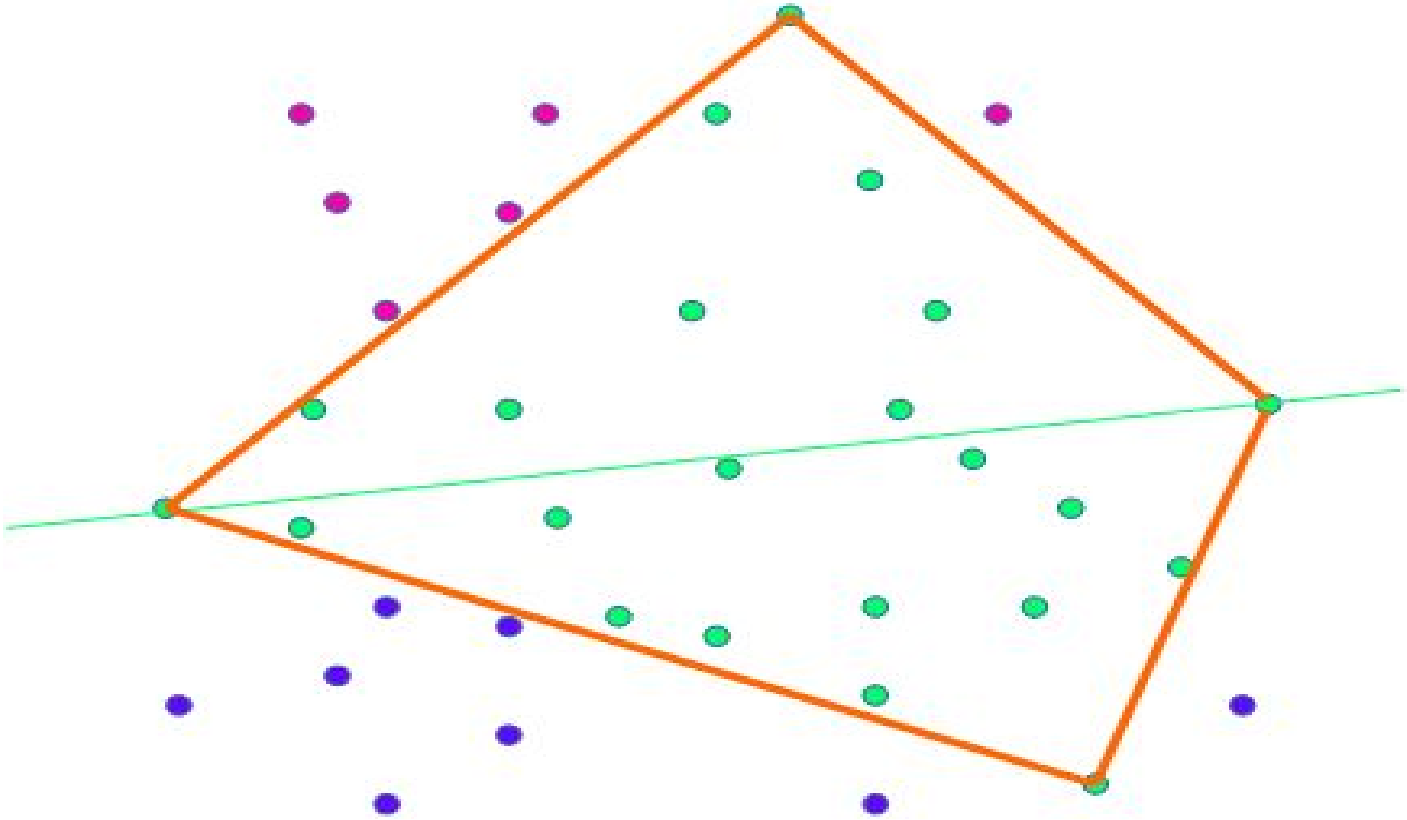
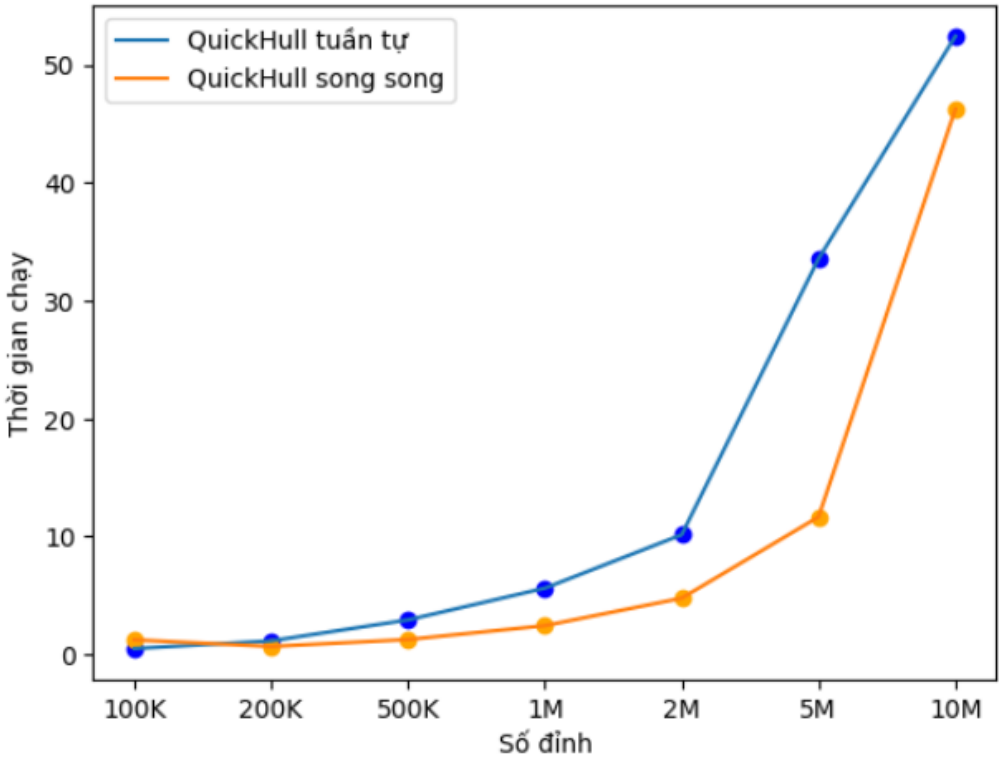


Figure 3: Line with P_{min} and P_{max}

Comparison



(a) Plot

Total points	Sequential Time(sec	Parallel Time(sec)
100K	0.5079820156	1.247114658
200K	1.162046909	0.6845638752
500K	2.936038733	1.275861979
1M	5.626057386	2.449432611
2M	10.15800214	4.77933383
5M	33.55099511	11.66812778
10M	52.41668677	46.23393369

(b) Result

Figure 4: Result of QuickHull and QuickHull using MultiProcessing

CudaCHPre2D [2]

- Another approach to accelerate the calculation of convex hulls is to filter the input points as a preprocessing stage
- Purpose: To reduce the input set P and transform the problem into a smaller one, without affecting the output result

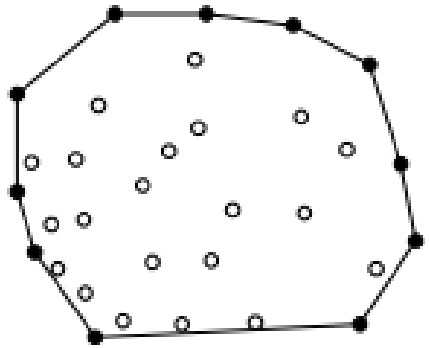
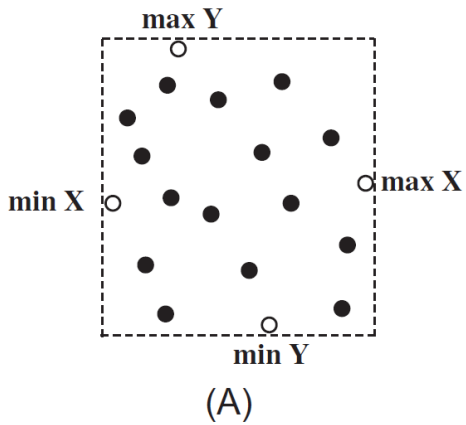


Figure 5: Interior points to be discarded

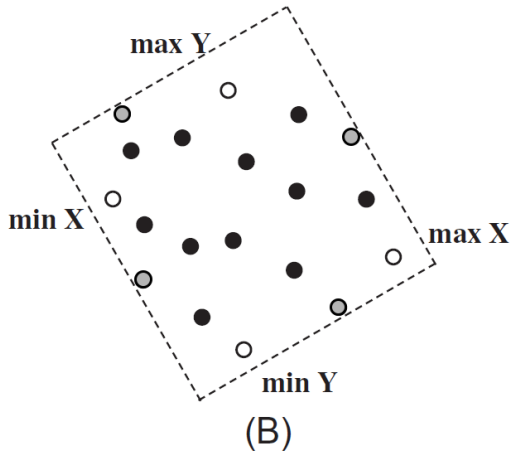
CudaCHPre2D

The basic ideas

- A convex hull is a unique polygon formed by a sequence of extreme points
- Extreme points are the highest/lowest/leftmost/rightmost points of P, or points with the largest/smallest x or y coordinates
- Rotating the set of points provides a new set of extreme points



(a) Original point set



(b) Rotated 30 degrees

CudaCHPre2D

Procedure

1. **Locate 16 extreme points**: 4 extreme points in each rotation: 0^o, 30^o, 45^o, 60^o
   - 3 kernels are designed for rotating the set of points
   - In each kernel, each thread is responsible for calculating the new and rotated position of a point
   - Find P_{minx}, P_{maxx}, P_{miny}, P_{maxy} using parallel reduction
2. **Form a convex polygon** of the 16 extreme points on the CPU
3. **Discard interior points** that locate in the convex polygon on a simple kernel
   - Each thread is responsible for checking a point to determine whether it falls in the convex polygon

Algorithm Design [3]

- The proposed GPU-accelerated convex hull algorithm is inspired by the filtering approach aforementioned.
- The procedure is composed of 2 main stages:
  - Two rounds of preprocessing performed on the GPU
  - The finalization of finding the convex hull on the CPU

Algorithm Design

The procedure is further broken down into 3 steps:

1. **Discard** points locating inside a quadrilateral formed by 4 extreme points P_{minx}, P_{maxx}, P_{miny}, P_{maxy}
2. **Distribute** the remaining points into 4 sub regions, sorting by x,y-coordinates
   Further discard interior points using a *Sorting-based Preprocessing Apporach (SPA)*
3. **Calculate** the convex hull of the simple polygon

Algorithm Design

Step 1: Points' distribution and the 1^{st} round of discarding

1. Find 4 extreme points P_{minx}, P_{maxx}, P_{miny}, P_{maxy} using parallel reduction
2. Determine the distribution of all points in parallel

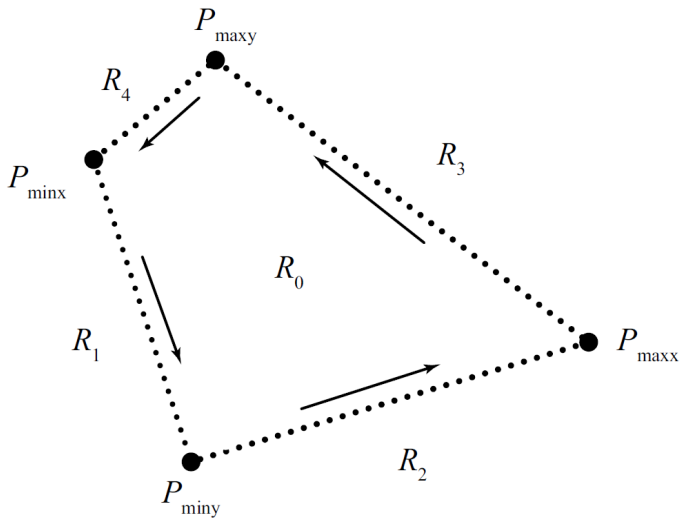


Figure 7: The distribution of points

3. Discard the points locating in region R_0

Algorithm Design

Step 2: The 2^{nd} round of discarding and forming simple polygon

1. Sort R_1, R_2, R_3, and R_4 separately in parallel

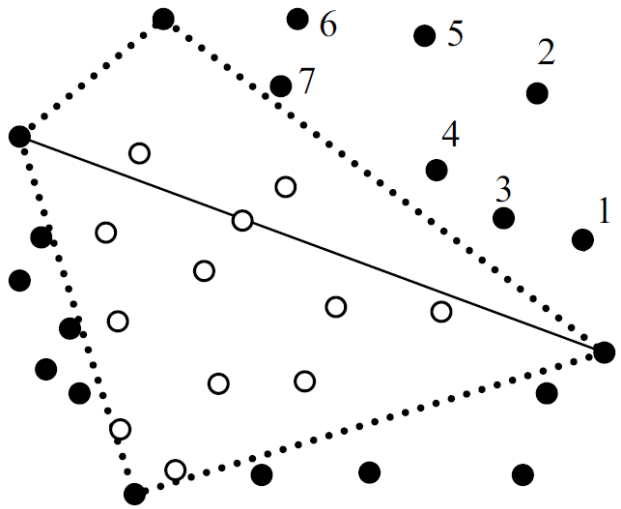


Figure 8: Demonstration of sorted points in the upper left region R_3. The rule: x in descending order, y in ascending order

Algorithm Design

Step 2: The 2^{nd} round of discarding and forming simple polygon

2. Perform the SPA to further discard interior points and simultaneously form a simple chain for the remaining points in each sub region

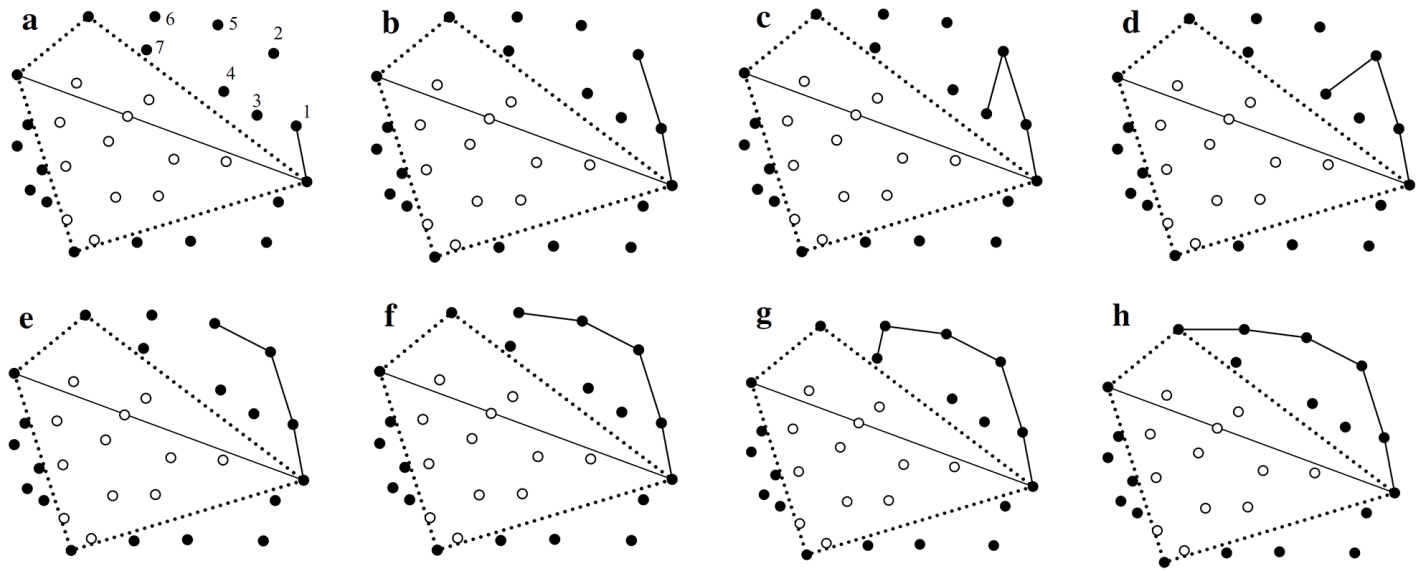
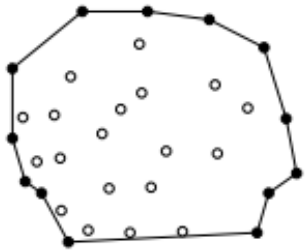


Figure 9: Example of forming the chain in R_3

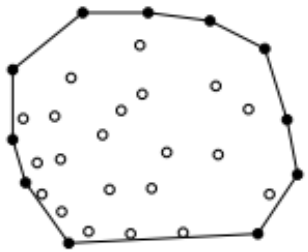
Algorithm Design

Step 3: Calculating the convex hull of the simple polygon

- The output of the previous step is a simple polygon, which is also an approximate convex hull.
- To calculate the exact convex hull of the input point set, Melkman's algorithm, which is considered to be the best convex hull algorithm for simple polygons, is chosen.
- The convex hull of the simple polygon is the convex hull of the input data set.



(a) A simple polygon



(b) The desired convex hull

Figure 10: The convex hull of a simple polygon

Dataset: Efficiency on the GTX 660M

- For small size of testing data, the Qhull is faster than the proposed CudaChain, while CudaChain is much faster than Qhull for the large size of testing data.
- CudaChain over Qhull become larger with the increasing of the data size. The speedup is about 3×–4× on average and 5×–6× in the best cases.

Dataset: Efficiency on the GTX 660M

Size	Qhull	CudaChain				Speedup
		Total	GPU	CPU	CPU(%)	
100k	27	42.5	39.6	2.9	6.82	0.64
200k	52	45.9	43.1	2.8	6.10	1.13
500k	124	65.6	61.4	4.2	6.40	1.89
1M	237	75.0	70.8	4.2	5.60	3.16
2M	426	129.1	123.2	5.9	4.57	3.30
5M	605	174.8	169.4	5.4	3.09	3.46
10M	1171	351.8	345.9	5.9	1.68	3.33
20M	2353	587.4	581.9	5.5	0.94	4.01

Figure 11: Comparison of running time (/ms) for point sets distributed in squares on GTX 660M

Dataset: Efficiency on the GTX 660M

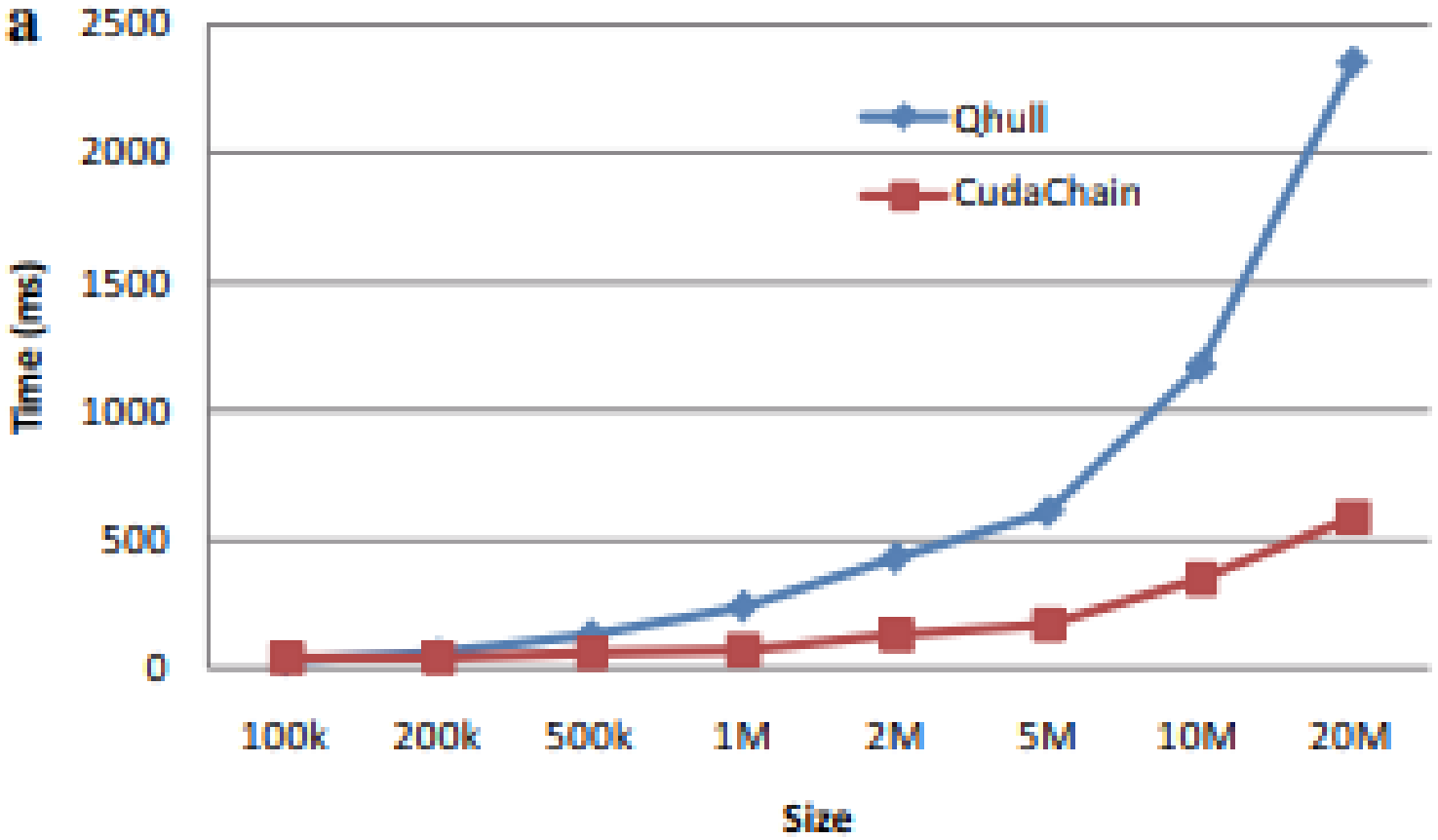


Figure 12: The efficiency of CudaChain on the GPU GTX 660M against CPU-based Qhull using the same datasets and the same machine for Point sets distributed in squares

Dataset: Efficiency on the GT 640

- For small size of testing data, the Qhull is also faster than the algorithm CudaChain, while CudaChain is much faster than Qhull for the large size of testing data.
- The speedups of CudaChain over Qhull also become larger with the increasing of the data sizes. The speedup is about 3×–4× on average and 4×–5× in the best cases.

Dataset: Efficiency on the GT 640

Size	Qhull	CudaChain				Speedup
		Total	GPU	CPU	CPU(%)	
100k	15	25.5	24.2	1.3	5.10	0.59
200k	16	29.1	27.9	1.2	4.12	0.55
500k	47	40.4	38.4	2.0	4.95	1.16
1M	109	46.9	44.6	2.3	4.90	2.32
2M	202	83.5	81.2	2.3	2.75	2.42
5M	515	147.0	144.5	2.5	1.70	3.50
10M	1034	321.9	319.7	2.2	0.68	3.21
20M	2215	544.4	541.5	2.9	0.53	4.07

Figure 13: Enter Caption

Application: Find location of the wifi station

- The ideal location is the centroid of the convex hull

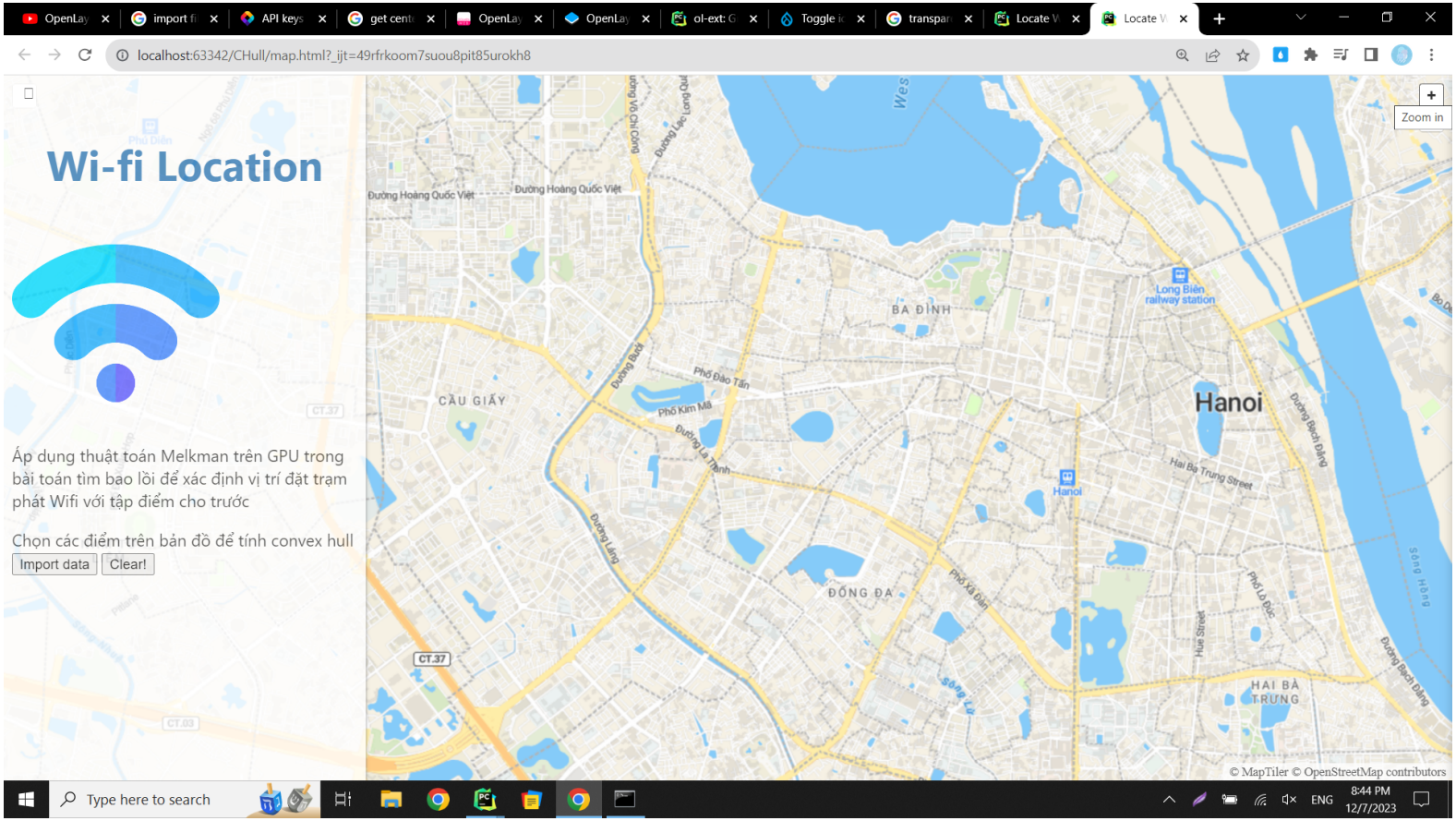


Figure 14: User Interface

Application: Find location of the wifi station

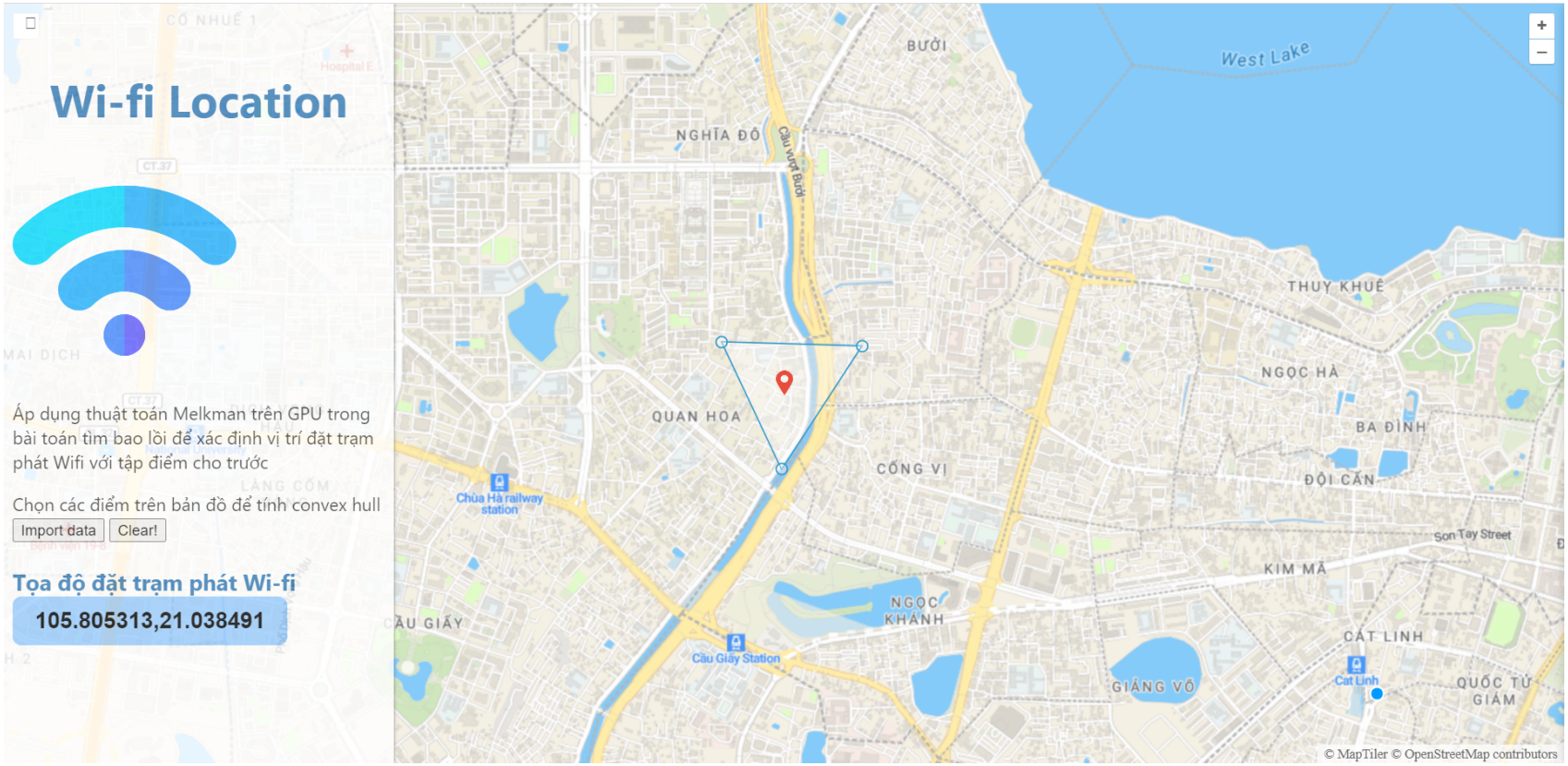


Figure 15: User Interface

Application: Find location of the wifi station

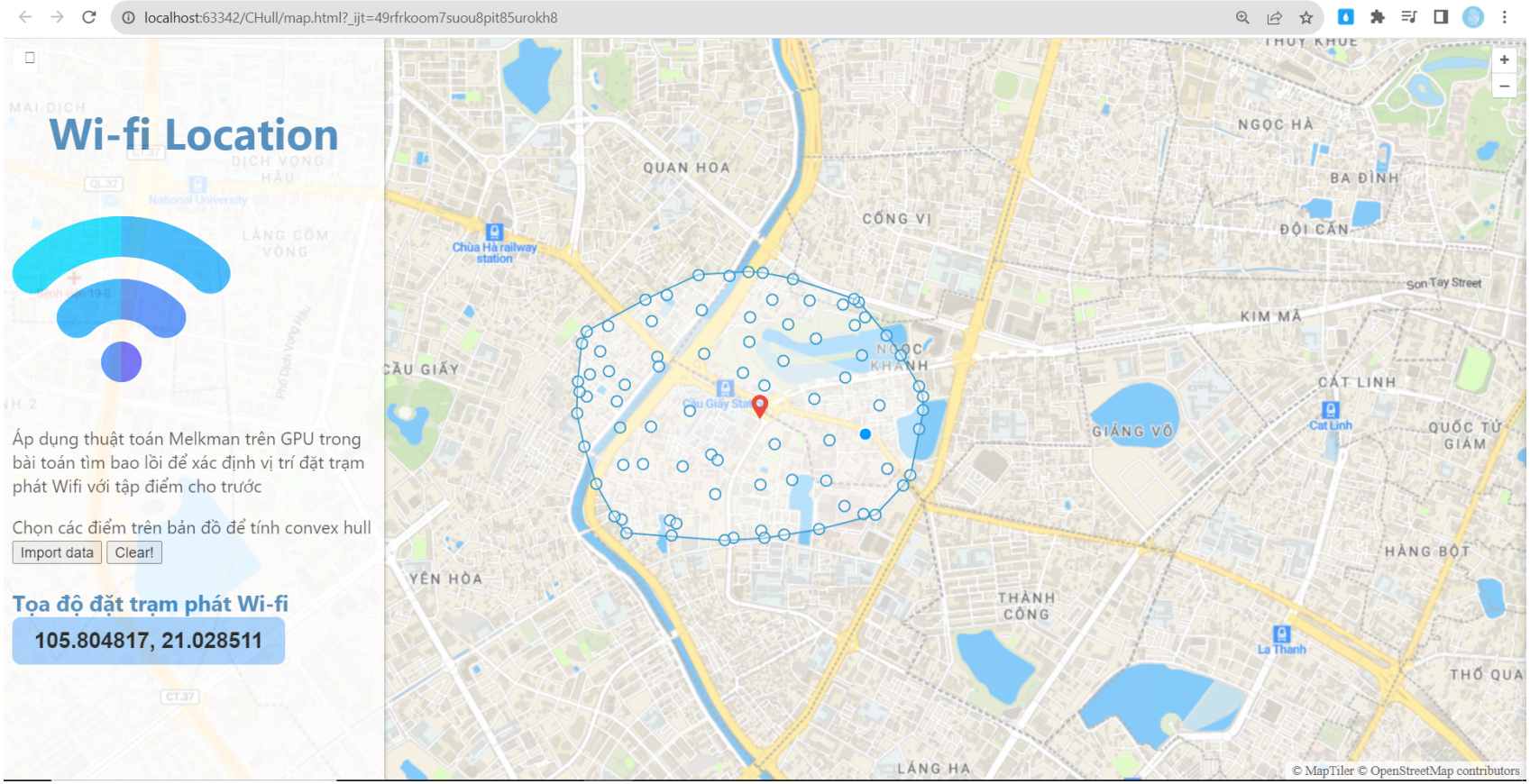


Figure 16: User Interface

References I

[1] Jiayin Zhang, Gang Mei, Nengxiong Xu, and Kunyang Zhao.

A novel implementation of quickhull algorithm on the GPU.

CoRR, abs/1501.04706, 2015.

[2] Jiayu Qin, Gang Mei, Salvatore Cuomo, Guo Sixu, and Yixuan Li.

Cudachpre2d: A straightforward preprocessing approach for accelerating 2d convex hull computations on the gpu.

Concurrency and Computation Practice and Experience, 32, 04 2019.

[3] Gang Mei.

Cudachain: A practical gpu-accelerated 2d convex hull algorithm.

CoRR, abs/1508.05488, 2015.

Thank you for your attention!