

# Voronoi diagram - A survey of various fast algorithms

Mingeon Jeong, Minseop Lee, Dongwon Lee, Chaewoon Lee

May 25, 2023

이 글은 Voronoi Diagram을  $O(n^2)$  또는  $O(n \log n)$ 의 시간에 구하는 알고리즘인 Fortune's Algorithm, Bowyer-Watson Algorithm, Jump Flooding Algorithm, Using divide and conquer, Incremental Algorithm을 소개한다.

## 1 Fortune's Algorithm

Fortune의 알고리즘은 포물선의 성질을 이용한 알고리즘이다. 포물선은 초점과의 거리와 준선과의 거리가 같은 집합의 모임이기 때문에, 초점이 각각  $P$ 와  $Q$ 이고 준선이 같은 선인 두 포물선의 교점은  $PQ$ 의 수직이등분선 위에 있다는 성질을 갖는다. 이를 이용하여  $O(n \log n)$ 에 Voronoi Diagram을 찾을 수 있다.

먼저 구체적인 알고리즘을 이야기하기 전에, 몇 가지 용어들을 정의하자.

- *sweep line*: 스위핑을 진행하는 직선으로  $x = x_0$ 로 쓰자. 아래 모든 과정에서 *sweep line*이  $x = x_0$ 이면  $x$ 좌표가  $x_0$  이하인 점들만 고려한다.
- *beach line*: 여러 개의 포물선의 일부분으로 이루어진 가장 오른쪽(가장  $x = x_0$ 와 가까운) 경계를 말한다. 준선은  $x = x_0$ 로 고정할 것이므로, 초점들만 리스트로 저장하면 *beach line*을 알 수 있다. 이 초점들의 목록을 Focus라고 하자. **Figure 1.1**은 *beach line*의 예시이다.
- *site event*: 점  $P_i$ 가 추가되는 이벤트이다. 이 이벤트가 실행되면, Focus에 점  $P_i$ 가 추가된다. 이 이벤트는  $x = x_0$ 인 점이 존재하면 실행된다.
- *circle event*: *beach line*을 이루는 어떤 포물선이 다른 포물선에 가려져 하나의 포물선이 사라지는 이벤트이다. 이 이벤트가 실행되면 Focus에 가려지는 포물선을 나타내는 점이 제거된다. 이 이벤트는 Focus에 포함되어 있으며  $y$ 좌표가 인접한 세 점의 외접원의 가장 오른쪽보다  $x = x_0$ 가 오른쪽에 있을 때 실행된다. 이 때 이 세 점의 외접원의 중심을 *circle point*라고 한다. **Figure 1.2**에 그 원과 *circle point*가 보여지고 있다.

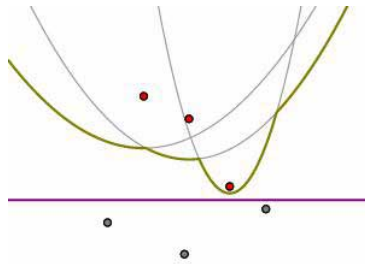


Figure 1.1

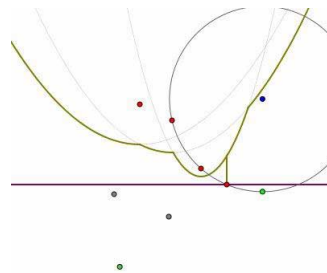


Figure 1.2

구체적인 알고리즘은 아래와 같다:

1. 점을  $x$ 좌표가 커지는 순서대로 정렬한다. 정렬 후  $i$ 번째 점을  $P_i$ 라 두자.
2. 양쪽과 연결된 양방향 연결리스트(Doubly Linked List) Focus를 만든다. 이 연결리스트는  $y$ 좌표가 증가하는 순서로 점들을 담는다.
3. *sweep line*  $x = x_0$ 에 대해  $x_0$ 을 증가시키면서
  - *site event*가 발생한 경우: Focus에 점  $P_i$ 를 넣는다. 이 과정에서 알맞은 위치를 찾기 위해  $y$ 좌표를 기준으로 이분 탐색(binary search)를 사용한다. 또한, 이 과정에서 새롭게 생긴 포물선과 기존의 *beach line*의 교점을 이어 Voronoi Diagram을 이루는 선분에 추가한다.
  - *circle event*가 발생한 경우: 점  $P_{i-1}, P_i, P_{i+1}$ 에서 *circle event*가 발생한 경우 Focus에서 점  $P_i$ 를 제거한다. 또한, 이 이후에는 두 개의 포물선과 교점이 생기므로, 그 교점과 *circle point*를 이어 2개의 선분을 추가한다. (하나는 기존과 겹칠 것이다)
4.  $x_0 = \infty$ 가 될 때까지 위 작업을 반복하여 얻은 선분들은 Voronoi Diagram을 이룬다.

다음 웹페이지에서 위 과정을 직접 확인할 수 있다: <http://www.eecs.tufts.edu/~vporok01/c163/>

## 2 Bowyer-Watson Algorithm

Bowyer-Watson은 Voronoi Diagram과 쌍대관계인 들로네 삼각분할을  $O(n^2)$ 의 시간복잡도에 해결하는 방법이다. 만약 Quad Tree를 사용할 경우  $O(n \log n)$ 의 시간복잡도로 구현할 수 있다. 이 알고리즘의 구현 방식은 아래와 같다:

1. 모든 점을 포함하는 가장 큰 정삼각형(*Supra-Triangle*)를 잡는다.
2.  $i$ 번째 점을  $P_i$ 라 두고, 점을 정해진 순서대로 하나씩 돈다.
  - 첫 번째 점의 경우: 위의 가장 큰 정삼각형(*Supra-Triangle*)과 연결한다.
  - 첫 번째 점이 아닌 경우: 이미 그린 삼각형들  $T_i$ 의 외접원들 중,  $P_i$ 를 포함하는 외접원을 만드는 삼각형  $T_i$ 들을 제거하고, 그 삼각형들의 꼭짓점들과  $P_i$ 를 이어 새로운 삼각형을 만든다.
3.  $N$ 개의 점에 대해 위 작업을 반복하고, 가장 큰 정삼각형(*Supra-Triangle*)을 지우면 들로네 삼각분할이 만들어진다.

Figure 2.1는 위 과정을 그림으로 그린 것이다.

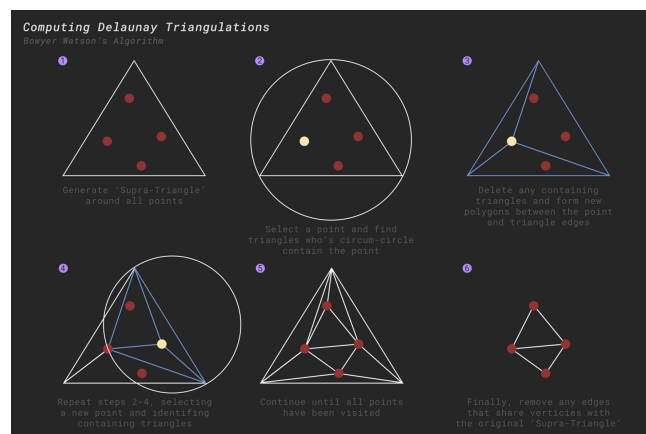


Figure 2.1

### 3 Jump Flooding Algorithm

Jump Flooding Algorithm은 가로  $N$ , 세로  $N$ 의 격자를 이용하여 Voronoi Diagram을 얻거나 Distance transforms를 하는데 사용되는 알고리즘이다. 시간복잡도는  $O(N^2 \log N)$ 이며 이 알고리즘의 구현 방식은 아래와 같다:

1. 처음 점의 좌표에 그 점의 색으로 마킹한다. 이렇게 처음 flooding의 시작이 되는 점을 *seed*라고 한다.
2.  $\text{step size}(k)$ 를  $N/2$ 부터 시작해 절반씩 줄이며 아래 수식에 따라 색을 마킹한다.  
 각  $P = (x, y)$ 에 대해  $Q = (x + i, y + j)$  where  $i, j \in -k, 0, k$ 인 모든  $Q$ 가
  - $Q$ 는 마킹되어 있지 않으나,  $P$ 는 마킹된 경우  $Q$ 의 색으로 마킹한다.
  - $Q$ 와  $P$  모두 마킹된 경우,  $Q$ 의 시드에 해당하는 것과  $P$ 의 시드에 해당하는 것 중 더 가까운 것으로 마킹한다.
3.  $O(\log N)$ 개의 점에 대해 위 작업을 반복하면 Voronoi Diagram이 만들어진다.

Figure 3.1는 위 과정을 그림으로 그린 것이다.

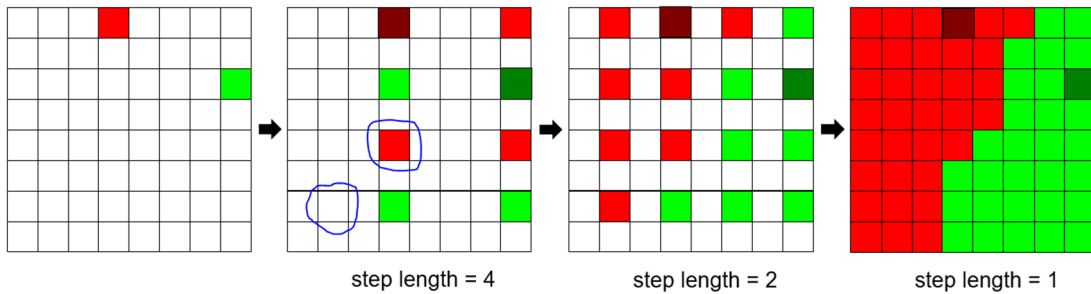


Figure 3.1

### 4 Using divide and conquer

분할 정복 알고리즘을 이용해 Voronoi Diagram을  $O(n \log n)$ 에 구할 수 있다.

1. 분할:  $x$ 좌표를 기준으로 양쪽에 각각 절반의 점이 존재하도록 나눈다.
2. 마지막: 점이 3개 이하인 경우 수식적으로 Voronoi Diagram을 구한다.
3. 병합: 경계의 점들을 *monotone chain*  $C$ 로 잇는다. 그 체인에 속하는 점을 체인 순서대로  $C_1, C_2, C_3, \dots, C_k$ 라 두었을 때  $C_i$ 와  $C_{i+1}$ 의 수직이등분선이 경계면의 Voronoi Diagram이 된다.

아래는 세 과정을 각각 나타낸 것이다.

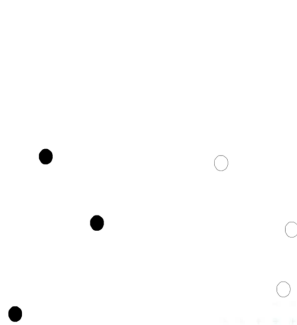


Figure 4.1

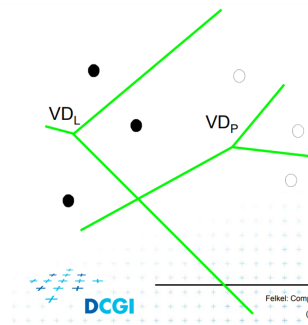


Figure 4.2

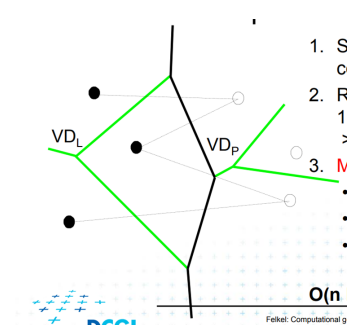


Figure 4.3

이제 *monotone chain*을 만드는 과정을 살펴보자.

1. 각각의 분할된 다이어그램에서 가장  $y$ 좌표가 큰 한 점을 고른다. 이 점을  $P_1, P_2$ 라 두자.
2.  $P_1$ 과  $P_2$ 를 수직이등분선하는 직선을 그은 뒤,  $P_1, P_2$ 의 셀의 경계와의 교점을 구한다.
3. 위에서 구한 교점들 중 가장  $y$ 좌표가 큰 교점을 선택하여, 그 지점까지 선분을 잇는다.
4. 위의 교점을 만든 수직이등분선을 만드는 두 점을 구한다. 즉, 그 두 점의 수직이등분선은  $y$ 좌표가 가장 큰 교점을 만든다. 이 두 점 중 하나는  $P_1$  또는  $P_2$ 일 것이므로, 겹치는 점을 제외한 나머지 두 점을 선택한다. 다시 2로 돌아가 과정을 반복한다.

*monotone chain*을 만들 때, 각 정점은 최대 1번 확인하고, 각 간선은 최대 2번 확인하므로 시간복잡도는  $O(n)$ 이다. 즉, 분할정복 과정의 전체 시간복잡도가  $O(n \log n)$ 이 된다. **Figure 4.4**는 위 과정을 그림으로 그린 것이다.

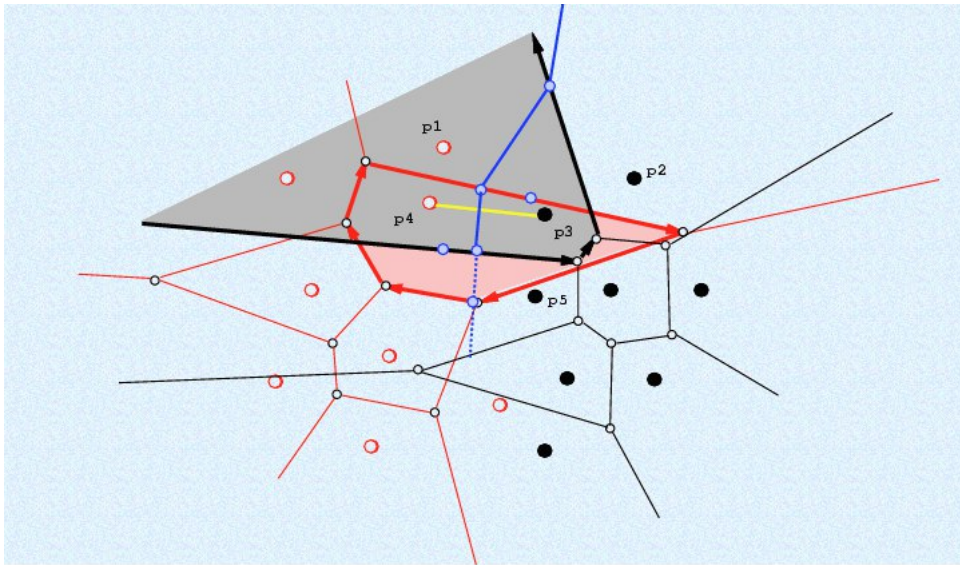


Рис. 4 Аналогично рисунку 3 обновляем один из концов отрезка. Пускаем луч и т.д

Figure 4.4

## 5 Incremental Algorithm

Incremental Algorithm을 이용하면  $O(n^2)$ 의 시간에 Voronoi Diagram을 구할 수 있다. 이 알고리즘의 기작은 분할 정복을 이용한 Voronoi Diagram 알고리즘에서 *monotone chain*을 구하는 것과 비슷하다. 알고리즘의 구현 방식은 아래와 같다:

1. 각 점  $P_i$ 에 대하여 다음을 반복한다.
  - $i = 1$ : 그 점의 셀을 전체 영역으로 잡는다.
  - $i \geq 2$ :  $P_i$ 가 포함된 셀을  $P_j$ 의 셀이라고 하자. 그러면 선분  $P_i P_j$ 의 수직이등분선은 Voronoi Diagram을 이루는 선이 된다. 이제 이 선과  $P_j$ 의 셀의 경계선과의 교점을 구하자. 그 교점에서 다시 셀을 찾고 수직이등분선을 긋는 것을 반복한다.
2.  $n$ 개의 점에 대해 위 작업이 종료되면 남은 선분들이 Voronoi Diagram이 된다.

위 과정에서 한번에 반복당 최대  $n$ 개의 점을 확인하기 때문에 시간복잡도는  $O(n^2)$ 이다. **Figure 5.1**는 위 과정을 그림으로 그린 것이다.

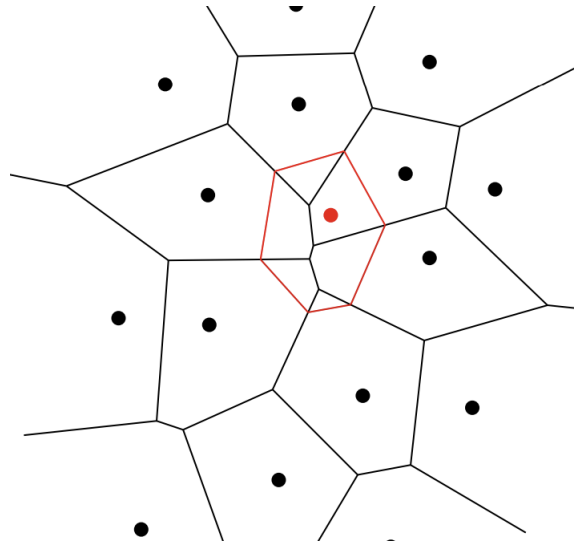


Figure 5.1

## 6 References

### 6.1 Fortune's Algorithm

- [1] [https://en.wikipedia.org/wiki/Fortune's\\_algorithm](https://en.wikipedia.org/wiki/Fortune's_algorithm)
- [2] <https://www.ams.org/publicoutreach/feature-column/fcarc-voronoi>
- [3] <https://kipl.tistory.com/7>

### 6.2 Bowyer-Watson Algorithm

- [4] <https://www.youtube.com/watch?v=GctAunEuHt4>

### 6.3 Jump Flooding Algorithm

- [5] <https://www.youtube.com/watch?v=GctAunEuHt4>
- [6] [https://en.wikipedia.org/wiki/Jump\\_flooding\\_algorithm](https://en.wikipedia.org/wiki/Jump_flooding_algorithm)

### 6.4 Using divide and conquer

- [7] [https://cw.fel.cvut.cz/b181/\\_media/courses/cg/lectures/06-voronoi-split.pdf](https://cw.fel.cvut.cz/b181/_media/courses/cg/lectures/06-voronoi-split.pdf)
- [8] <https://sudonull.com/post/76327-Building-a-Voronoi-diagram-using-the-divide-and-conquer>

### 6.5 Incremental Algorithm

- [9] <https://www.ic.unicamp.br/~rezende/ensino/mo619/Sacristan,%20Voronoi%20Diagrams.pdf>