효율적이고 빠른 격자 보르노이 다이어그램 계산

이동원 (서울과학고 2년), 이민섭 (서울과학고 2년), 이채운 (서울과학고 2년), 정민건 (서울과학고 2년)
지도교사 : 윤상호 (서울과학고 수학 전공), 지도교수 : 김준석 (고려대학교 수학과)

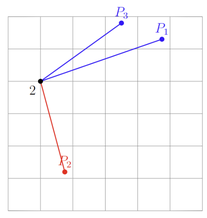
연구배경 및 목적

- 보로노이 다이어그램은 공간 분석에 있어 중요한 도구로, 근접 관계, 자원 배분 및 가장 가까운 point를 찾는 데 사용된다. 이 중 격자에서의 보로노이 다이어그램의 경우 특히 이미지 처리, 지형 모델링 등에서 활발하게 사용되나, 고해상도의 격자와 많은 수의 격자점으로 인해 전통적인 계산 방식으로는 시간과 메모리 측면에서 한계가 있다.] 또한 격자에서의 보로노이 다이어그램을 구하는 방법은 일반적인 보로노이 다이어그램을 구하는 방법과는 차이가 있어 정확하고 효율적으로 격자 보로노이 다이어그램을 구하는 알고리즘이 필요하다.
- 본 연구에서는 격자 보로노이 다이어그램을 구하는 알고리즘을 다양한 방법으로 제시하여 그 성능을 비교하고, 나아가 각 알고리즘을 조합하여 격자의 크기와 격자점의 수에 관계없이 효율적으로 격자 보로노이 다이어그램을 구하는 방법을 제시하고자 한다.

연구방법

1. 문제 정의

격자 보로노이 다이어그램을 구하는 구체적인 목표는 다음과 같다. MxM 격자에서 N개의 시드점이 주어졌을 때, 각 격자에 대해 N개의 시드점 중 가장 가까운 점을 찾아내고 이를 저장하는 알고리즘을 작성하는 것이다. 오른쪽 그림에서, P1 P2 P3 중 2가 적힌 격자점에 가장 가까운 시드점은 P2이므로, 이 격자점에는 2를 저장하게 된다.

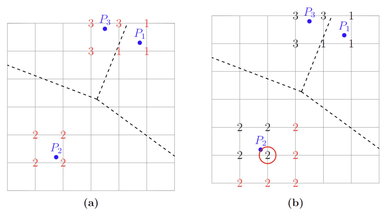


2. 연구 과정

본 연구에서는 격자에서의 보로노이 다이어그램을 구하기 위해 가장 간단한 알고리즘인 Naive 알고리즘을 포함해 총 7개의 알고리즘을 고안하였다. 각 알고리즘의 실행시간을 확인해보기 위해 Polygon을 이용해 격자점 및 시드점의 개수에 따라 실행시간을 측정한 데이터를 수집한 후, 이를 그래프로 나타내어 비교하였다. 또한,Sparse 알고리즘을 제외한 알고리즘들의 이론적인 시간복잡도를 유도한 후, 실제 실행시간을 측정한 데이터로 선형회귀를 하여 시간복잡도 식의 정확한 계수를 구하였다.

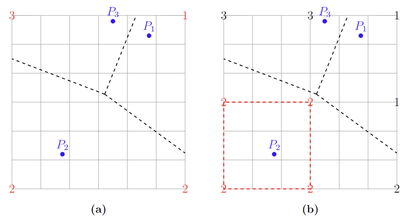
3. 각 알고리즘의 figure 및 간략한 설명

Dijkstra



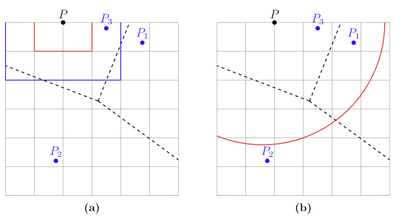
다익스트라 알고리즘의 아이디어를 응용하여 현재까지 가장 seed와 가까운 점에서 8방향으로 업데이트 후 예외 발생 확률을 줄이기 위해 두 번째로 가까운 seed까지 저장해 업데이트한다.

Sparse



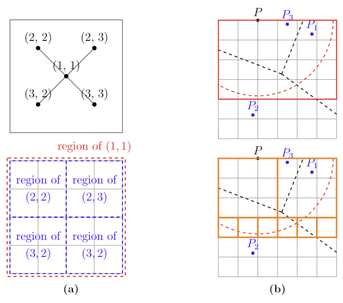
(a)와 같이 전체 격자의 네 꼭짓점에서 가장 가까운 시드점을 찾아 번호를 부여한다. 이후 격자를 (b)와 같이 4분할하여 꼭짓점에 번호를 부여하는 과정을 반복한다. 이때, 분할된 격자에서 네 꼭짓점의 번호가 모두 같다면 그 내부의 격자점에 대해서는 모두 같은 번호를 부여할 수 있다.

Dense



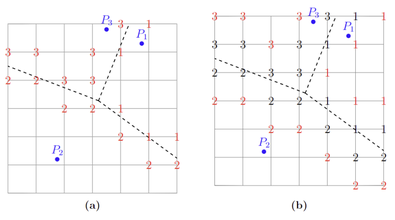
한 점 P를 중심으로 임의의 시드점을 포함하는 정사각형의 한 변의 길이의 최솟값 2k를 이분탐색을 통해 찾는다. 이때 P와 가장 가까운 시드점과 P 사이의 거리는 $\sqrt{2}(k+1)$ 이하이므로 이 반경 내에서의 시드점에 대해서만 검사하면 가장 가까운 시드점을 찾을 수 있다.

Quad



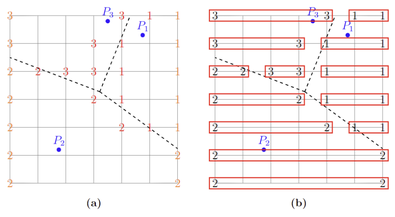
dense 알고리즘에서 세그먼트 트리와 머지 소트 트리의 아이디어를 이용해 quad 트리를 만든다. 이후 분할된 영역에 포함되는 seed의 번호를 저장해 빠른 시간에 범위에 포함되는 seed들의 목록을 구한다.

BFS



(a)와 같이 보로노이 직선의 인접한 격자점에 그 점과 가장 가까운 시드점의 번호를 부여한다. 이후 Breadth-First Search(BFS) 기법을 이용해 (b)와 같이 나머지 격자점들에 시드점의 번호를 부여한다.

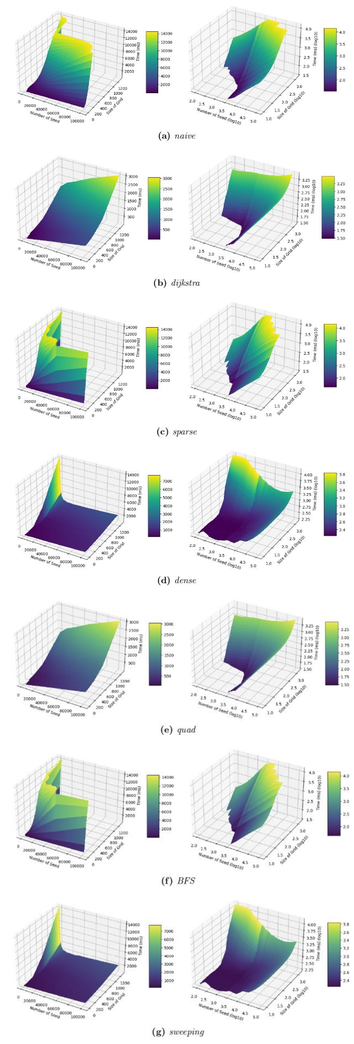
Sweeping



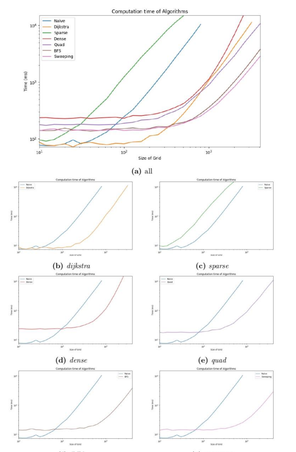
(a)와 같이 보로노이 직선의 인접한 격자점 및 행의 양 끝점에 그 점과 가장 가까운 시드점의 번호를 부여한다. 이후 행의 왼쪽 끝에서 시작하여 오른쪽 끝으로 한 칸씩 이동하면서 다른 시드점의 번호가 나올때까지 왼쪽 끝 격자점의 번호를 부여한다. 이 과정을 반복하여 오른쪽 끝까지 번호를 부여한다.

결과

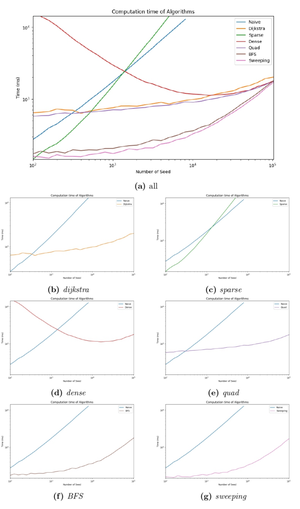
그래프 1. 격자의 크기와 시드점의 개수에 따른 실행시간의 3D-plot



그래프 2.격자의 크기에 따른 실행시간의 2D-plot



그래프 3. 시드점의 개수에 따른 실행시간의 2D-plot



각 알고리즘의 시간 복잡도

- *Dijkstra*

$$T = 79.69 + 1.975\times10^{-5}\cdot N\sqrt{N} + 5.369\times10^{-4}\cdot M^2\log M \quad R^2 = 0.9795 \quad (N < M^2)$$
$$T = 6.25 + 6.932\times10^{-3}\cdot N + 3.483\times10^{-4}\cdot M^2\log M \quad R^2 = 0.9994 \quad (N > M^2)$$

- *Sparse*

Not Found $(N < M^2)$
$$T = 153.74 + 1.332\times10^{-5}\cdot NM^2 \quad R^2 = 0.9885 \quad (N > M^2)$$

- *Dense*

$$T = 318.73 + 1.873\times10^{-6}\cdot\frac{M^4}{N} + 1.213\times10^{-4}\cdot M^2\log M \quad R^2 = 0.9848$$

- *Quad*

$$T = 108.49 + 8.343\times10^{-3}\cdot N + 1.331\times10^{-5}\cdot M^2\log^2 M \quad R^2 = 0.9703$$

- *BFS*

$$T = 20.82 + 1.311\times10^{-3}\cdot N\log N + 3.907\times10^{-7}\cdot NM + 1.961\times10^{-4}\cdot M^2 \quad R^2 = 0.9985$$

- *Sweeping*

$$T = 26.63 + 1.294\times10^{-3}\cdot N\log N + 5.912\times10^{-7}\cdot NM + 1.578\times10^{-4}\cdot M^2 \quad R^2 = 0.9982$$

논의

Dijkstra 알고리즘은 그래프 2에서 M에 따른 기본 전처리 시간복잡도는 naïve 알고리즘 정도로 훨씬 작게 나타난다. 그러나 M과 연관된 큰 상수시간으로 인해 다른 알고리즘에 비해 시간복잡도가 크게 나타난다. 또한 이 알고리즘은 다른 알고리즘에 비해 N에 대한 기울기가 최소이므로 N에 영향을 크게 받지 않아 특히 M이 작고 N이 큰 경우에 유용하다.

Sparse알고리즘은 그래프 2에서 m에 따른 시간복잡도는 naïve 알고리즘과 유사한 그래프를 보여주지만 m이 커질 수록 naïve그래프의 시간복잡도가 sparse알고리즘에 비해 큰 결과를 보인다. 또한 그래프 3에서 N에 따른 시간복잡도는 상수항은 0에 가장 가까우며 이는 함수 호출 횟수가 N에 큰 영향을 받는다는 것을 나타낸다. 따라서 이 알고리즘은 N에 더 높은 정도를 보이고 M에서 더 낮은 정도의 시간복잡도를 보인다.

Dense알고리즘은 그래프 2에서 M에 대해 폭발적으로 증가하며 이는 N이 10^4시점에서 다른 알고리즘에 비해 M에 대한 의존성이 중요해지기 시작함을 나타낸다. 또한 그래프 3에서 N에 따른 시간복잡도는 N이 감소함에 따라 시간이 특이하게 증가하며 이는 이론적 분석의 결과와 일치한다.

Quad알고리즘은 그래프 2에서 N에 대한 전처리 시간과 M에 대한 시간 증가 모두 상당히 짧은 결과를 보여준다.

BFS알고리즘 및 sweeping알고리즘은 둘 다 보로노이 다이어그램을 계산하는 특징이 있다. 그래프 2에서 보로노이 다이어그램의 계산 시간으로 인해 N에 대한 시간복잡도가 dijkstra알고리즘에 비해 높은 결과를 보여주는 것을 알려준다.

결론

본 연구에서는 기존의 보로노이 다이어그램을 생성하는 알고리즘과 달리 그리드 기반 보로노이 다이어그램에 대한 빠른 알고리즘을 개발하는데 중점을 두었다. 그리드 기반 보로노이 다이어그램 알고리즘의 목적은 평면의 그리드로 분할하고 각 그리드에 가장 가까운 시드를 배정하는 것이다. 그 결과 가장 간단한 naive 알고리즘을 포함한 총 7개의 알고리즘을 개발하고 그리드 수와 시드의 개수를 기반으로 수행 시간을 비교하였다. 또한 sparse 알고리즘을 제외한 모든 알고리즘에 대해 이론적인 시간복잡도 함수를 도출하고 실제 실행 데이터를 사용한 선형회귀를 통해 시간복잡도 함수의 정확한 계수를 결정했다.

그 결과 일반적인 경우에는 BFS와 Sweeping이 빠른 속도를 보여주었으나, 격자 크기와 시드점 수가 적은 경우에는 Sparse, 격자의 크기가 작고 시드점의 수가 클 경우에는 Dijkstra와 Quad가 더 빠른 속도를 보여주었다.

참고문헌

[1] https://dl.acm.org/doi/abs/10.1145/116873.116880 ₩newline

[2] Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR), 23(3), 345-405. ₩newline

[3] Lau, B., Sprunk, C., ₩& Burgard, W. (2010, October). Improved updating of Euclidean distance maps and Voronoi diagrams. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 281-286). IEEE. ₩newline

[4] Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06. I3D '06. Redwood City, California: Association for Computing Machinery. pp. 109-116. ₩newline

[5] https://dl.acm.org/doi/pdf/10.1145/327070.327215 ₩newline